NAG Toolbox for MATLAB

e04vh

1 Purpose

e04vh solves sparse nonlinear programming problems.

2 Syntax

```
[x, xstate, xmul, f, fstate, fmul, ns, ninf, sinf, cw, iw, rw, user,
ifail] = e04vh(start, objadd, objrow, prob, usrfun, iafun, javar, nea,
a, igfun, jgvar, neg, xlow, xupp, xnames, flow, fupp, fnames, x, xstate,
f, fstate, fmul, ns, cw, iw, rw, 'nf', nf, 'n', n, 'nxname', nxname,
'nfname', nfname, 'lena', lena, 'leng', leng, 'lencw', lencw, 'leniw',
leniw, 'lenrw', lenrw, 'user', user)
```

Before calling e04vh, or one of the option setting routines e04vl, e04vm or e04vn, routine e04vg **must** be called.

3 Description

e04vh is designed to minimize a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimization, as well as for general nonlinear programs of the form

$$\underset{x}{\text{minimize}}\, f_0(x) \qquad \text{subject to } l \leq \begin{pmatrix} x \\ f(x) \\ A_L x \end{pmatrix} \leq u, \tag{1}$$

where x is an n-vector of variables, l and u are constant lower and upper bounds, $f_0(x)$ is a smooth scalar objective function, A_L is a sparse matrix, and $f(x)$ is a vector of smooth nonlinear constraint functions $\{f_i(x)\}$. The optional parameter **Maximize** specifies that $f_0(x)$ should be maximized instead of minimized.

Ideally, the first derivatives (gradients) of $f_0(x)$ and $f_i(x)$ should be known and coded by you. If only some of the gradients are known, e04vh estimates the missing ones by finite differences.

If $f_0(x)$ is linear and $f(x)$ is absent, (1) is a linear program (LP) and e04vh applies the primal simplex method (see Dantzig (1963)). Sparse basis factors are maintained by LUSOL (see Gill *et al.* (1987)) as in MINOS (see Murtagh and Saunders (1995)).

If only the objective is nonlinear, the problem is linearly constrained (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). For both nonlinear cases, e04vh applies a sparse sequential quadratic programming (SQP) method (see Gill *et al.* (2002)), using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for step-length control is an augmented Lagrangian, as in the dense SQP solver e04wd (see Gill *et al.* (1986c) and Gill *et al.* (1992)).

e04vh is suitable for nonlinear problems with thousands of constraints and variables, and is most efficient if only some of the variables enter nonlinearly, or there are relatively few degrees of freedom at a solution (i.e., many constraints are active). However, there is no limit on the number of degrees of freedom.

e04vh allows linear and nonlinear constraints and variables to be entered in an *arbitrary order*, and uses one (sub)program to define all the nonlinear functions.

The optimization problem is assumed to be in the form

$$\underset{x}{\text{minimize}}\, F_{\text{obj}}(x) \qquad \text{subject to } l_x \leq x \leq u_x, \qquad l_F \leq F(x) \leq u_F, \tag{2}$$

where the upper and lower bounds are constant, $F(x)$ is a vector of smooth linear and nonlinear constraint functions $\{F_i(x)\}$, and $F_{\text{obj}}(x)$ is one of the components of F to be minimized, as specified by the input parameter **objrow**. e04vh reorders the variables and constraints so that the problem is in the form (1).

Upper and lower bounds are specified for all variables and functions. The jth constraint may be defined as an equality by setting $l_j = u_j$. If certain bounds are not present, the associated elements of l or u should be set to special values that are treated as $-\infty$ or $+\infty$. Free variables and free constraints ('free rows') have both bounds infinite.

In general, the components of F are *structured* in the sense that they are formed from sums of linear and nonlinear functions of just some of the variables. This structure can be exploited by e04vh.

In many cases, the vector $F(x)$ is a sum of linear and nonlinear functions. e04vh allows these terms to be specified separately, so that the linear part is defined just once by the input arguments **iafun**, **javar** and **a**. Only the nonlinear part is recomputed at each x.

Suppose that each component of $F(x)$ is of the form

$$F_i(x) = f_i(x) + \sum_{j=1}^{n} A_{ij}x_j,$$

where $f_i(x)$ is a nonlinear function (possibly zero) and the elements A_{ij} are constant. The nf by n Jacobian of $F(x)$ is the sum of two sparse matrices of the same size: $F'(x) = G(x) + A$, where $G(x) = f'(x)$ and A is the matrix with elements $\{A_{ij}\}$. The two matrices must be *non-overlapping* in the sense that each element of the Jacobian $F'(x) = G(x) + A$ comes from $G(x)$ or A, but *not both.* The element cannot be split between $G(x)$ and A.

For example, the function

$$F(x) = \begin{pmatrix} 3x_1 + e^{x_2}x_4 + x_2^2 + 4x_4 - x_3 + x_5 \\ x_2 + x_3^2 + \sin x_4 - 3x_5 \\ x_1 - x_3 \end{pmatrix}$$

can be written as

$$F(x) = f(x) + Ax = \begin{pmatrix} e^{x_2}x_4 + x_2^2 + 4x_4 \\ x_3^2 + \sin x_4 \\ 0 \end{pmatrix} + \begin{pmatrix} 3x_1 - x_3 + x_5 \\ x_2 - 3x_5 \\ x_1 - x_3 \end{pmatrix},$$

in which case

$$F'(x) = \begin{pmatrix} 3 & e^{x_2}x_4 + 2x_2 & -1 & e^{x_2} + 4 & 1 \\ 0 & 1 & 2x_3 & \cos x_4 & -3 \\ 1 & 0 & -1 & 0 & 0 \end{pmatrix}$$

can be written as $F'(x) = f'(x) + A = G(x) + A$, where

$$G(x) = \begin{pmatrix} 0 & e^{x_2}x_4 + 2x_2 & 0 & e^{x_2} + 4 & 0 \\ 0 & 0 & 2x_3 & \cos x_4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} 3 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 & -3 \\ 1 & 0 & -1 & 0 & 0 \end{pmatrix}.$$

Note: the element $e^{x_2} + 4$ of $F'(x)$ appears in $G(x)$ and is not split between $G(x)$ and A although it contains a linear term.

The nonzero elements of A and G are provided to e04vh in co-ordinate form. The elements of A are entered as triples (i,j,A_{ij}) in the arrays **iafun**, **javar** and **a**. The sparsity pattern G is entered as pairs (i,j) in the arrays **igfun** and **jgvar**. The corresponding entries G_{ij} (any that are known) are assigned to appropriate array elements $\mathbf{g}(k)$ in user-supplied (sub)program **usrfun**.

The elements of A and G may be stored in any order. Duplicate entries are ignored. **igfun** and **jgvar** may be defined automatically by (sub)program e04vj when **Derivative Option** $= 0$ is specified and user-supplied (sub)program **usrfun** does not provide any gradients.

Throughout this document the symbol ϵ is used to represent the ***machine precision*** (see x02aj).

e04vh is based on SNOPTA, which is part of the SNOPT package described in Gill *et al.* (2005).

4 References

Dantzig G B (1963) *Linear Programming and Extensions* Princeton University Press

Eldersveld S K (1991) Large-scale sequential quadratic programming algorithms *PhD Thesis* Department of Operations Research, Stanford University, Stanford

Fourer R (1982) Solving staircase linear programs by the simplex method *Math. Programming* **23** 274–313

Gill P E, Murray W and Saunders M A (2002) *SNOPT: An SQP Algorithm for Large-scale Constrained Optimization* **12** 979–1006 SIAM J. Optim.

Gill P E, Murray W and Saunders M A (2005) Users' guide for SQOPT 7: a Fortran package for large-scale linear and quadratic programming *Report NA 05-1* Department of Mathematics, University of California, San Diego URL: ftp://www.cam.ucsd.edu/pub/peg/reports/sqdoc7.pdf

Gill P E, Murray W and Saunders M A (2005) Users' guide for SNOPT 7.1: a Fortran package for large-scale linear nonlinear programming *Report NA 05-2* Department of Mathematics, University of California, San Diego URL: ftp://www.cam.ucsd.edu/pub/peg/reports/sndoc7.pdf

Gill P E, Murray W, Saunders M A and Wright M H (1986c) Users' guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming *Report SOL 86-2* Department of Operations Research, Stanford University

Gill P E, Murray W, Saunders M A and Wright M H (1987) Maintaining *LU* factors of a general sparse matrix *Linear Algebra and its Applics.* **88/89** 239–270

Gill P E, Murray W, Saunders M A and Wright M H (1992) Some theoretical properties of an augmented Lagrangian merit function *Advances in Optimization and Parallel Computing* (ed P M Pardalos) 101–128 North Holland

Hock W and Schittkowski K (1981) *Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems* **187** Springer–Verlag

Murtagh B A and Saunders M A (1978) Large-scale linearly constrained optimization *14* 41–72 Math. Program.

Murtagh B A and Saunders M A (1982) A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints *Math. Program. Stud.* **16** 84–118

Murtagh B A and Saunders M A (1995) MINOS 5.4 Users' Guide *Report SOL 83-20R* Department of Operations Research, Stanford University

5 Parameters

5.1 Compulsory Input Parameters

1: **start – int32 scalar**

Indicates how a starting point is to be obtained.

start $= 0$

Requests that the Crash procedure be used, unless a Basis file is provided via optional parameters **Old Basis File**, **Insert File** or **Load File**.

start $= 1$

Is the same as **start** $= 0$ but is more meaningful when a Basis file is given.

start $= 2$

Means that **xstate** and **fstate** define a valid starting point (probably from an earlier call, though not necessarily).

Constraint: **start** $= 0$, 1 or 2.

2: **objadd – double scalar**

Is a constant that will be added to the objective row F_{obj} for printing purposes. Typically, **objadd** = 0.0D+0.

3: **objrow – int32 scalar**

Says which row of $F(x)$ is to act as the objective function. If there is no such row, set **objrow** $= 0$. Then e04vh will seek a feasible point such that $l_F \leq F(x) \leq u_F$ and $l_x \leq x \leq u_x$.

Constraint: $1 \leq$ **objrow** $\leq$ **nf** or **objrow** $= 0$ (or a feasible point problem).

4: **prob – string**

Is an 8-character name for the problem. **prob** is used in the printed solution and in some routines that output Basis files. A blank name may be used.

5: **usrfun – string containing name of m-file**

usrfun must define the nonlinear portion $f(x)$ of the problem functions $F(x) = f(x) + Ax$, along with its gradient elements $G_{ij}(x) = \dfrac{\partial f_i(x)}{\partial x_j}$. This (sub)program is passed to e04vh as the external parameter **usrfun**. (A dummy (sub)program is needed even if $f \equiv 0$ and all functions are linear.)

In general, **usrfun** should return all function and gradient values on every entry except perhaps the last. This provides maximum reliability and corresponds to the default option setting, **Derivative Option** $= 1$.

The elements of $G(x)$ are stored in the array **g**$(1:$ **leng**$)$ in the order specified by the input arrays **igfun** and **jgvar**.

In practice it is often convenient *not* to code gradients. e04vh is able to estimate them by finite differences, using a call to **usrfun** for each variable x_j for which some $\dfrac{\partial f_i(x)}{\partial x_j}$ needs to be estimated. However, this reduces the reliability of the optimization algorithm, and it can be very expensive if there are many such variables x_j.

As a compromise, e04vh allows you to code *as many gradients as you like*. This option is implemented as follows. Just before **usrfun** is called, each element of the derivative array **g** is initialized to a specific value. On exit, any element retaining that value must be estimated by finite differences.

Some rules of thumb follow:

(i) for maximum reliability, compute all gradients;

(ii) if the gradients are expensive to compute, specify optional parameter **Nonderivative Linesearch** and use the value of the input parameter **needg** to avoid computing them on certain entries. (There is no need to compute gradients if **needg** $= 0$ on entry to **usrfun**.);

(iii) if not all gradients are known, you must specify **Derivative Option** $= 0$. You should still compute as many gradients as you can. (It often happens that some of them are constant or zero.);

(iv) again, if the known gradients are expensive, don't compute them if **needg** $= 0$ on entry to **usrfun**;

(v) use the input parameter **status** to test for special actions on the first or last entries.

(vi) while **usrfun** is being developed, use the optional parameter **Verify Level** to check the computation of gradients that are supposedly known.

(vii)
usrfun is not called until the linear constraints and bounds on x are satisfied. This helps confine x to regions where the functions $f_i(x)$ are likely to be defined. However, be aware of the optional

parameter **Minor Feasibility Tolerance** if the functions have singularities on the constraint boundaries.

(viii)
set **status** $= -1$ if some of the functions are undefined. The linesearch will shorten the step and try again.

(ix) set **status** ≤ -2 if you want e04vh to stop.

Its specification is:

```
[status, f, g, user] = usrfun(status, n, x, needf, nf, f, needg,
leng, g, user)
```

Input Parameters

1: **status – int32 scalar**

Indicates the first and last calls to **usrfun**.

status $= 0$

There is nothing special about the current call to **usrfun**.

status $= 1$

e04vh is calling your (sub)program for the *first* time. You may wish to do something special such as read data from a file.

status ≥ 2

e04vh is calling your (sub)program for the *last* time. This parameter setting allows you to perform some additional computation on the final solution.

status $= 2$

The current **x** is *optimal*.

status $= 3$

The problem appears to be infeasible.

status $= 4$

The problem appears to be unbounded.

status $= 5$

An iterations limit was reached.

If the functions are expensive to evaluate, it may be desirable to do nothing on the last call. The first executable statement could be

```
if (status ≥ 2)
  return;
end
```

May be used to indicate that you are unable to evaluate f or its gradients at the current x. (For example, the problem functions may not be defined there.)

During the linesearch, $f(x)$ is evaluated at points $x = x_k + \alpha p_k$ for various step lengths α, where $f(x_k)$ has already been evaluated satisfactorily. For any such x, if you set **status** $= -1$, e04vh will reduce α and evaluate f again (closer to x_k, where $f(x_k)$ is more likely to be defined).

If for some reason you wish to terminate the current problem, set **status** ≤ -2.

2: **n – int32 scalar**

n, the number of variables, as defined in the call to e04vh.

3: **x(n) – double array**

The variables x at which the problem functions are to be calculated. The array x must not be altered.

4: **needf – int32 scalar**

5: **nf – int32 scalar**

6: **f(nf) – double array**

Concerns the calculation of $f(x)$.

nf is the length of the full vector $F(x) = f(x) + Ax$ as defined in the call to e04vh.

needf indicates if **f** must be assigned during this call of **usrfun**.

needf $= 0$

> **f** is not required and is ignored.

needf > 0

> The components of $f(x)$ corresponding to the nonlinear part of $F(x)$ must be calculated and assigned to **f**.
>
> If $F_i(x)$ is linear and completely defined by the ith row of A, A_i', then the associated value $f_i(x)$ is ignored and need not be assigned. However, if $F_i(x)$ has a nonlinear portion $f_i(x)$ that happens to be zero at x, then it is still necessary to set $f_i(x) = 0$. If the linear part A_i' of a nonlinear $F_i(x)$ is provided using the arrays **iafun**, **javar** and **a**, then it must not be computed again as part of $f_i(x)$.

To simplify the code, you may ignore the value of **needf** and compute $f(x)$ on every entry to **usrfun**.

needf may also be ignored with **Derivative Linesearch** and **Derivative Option** $= 1$. In this case, **needf** is always 1, and **f** must always be assigned.

f contains the computed functions $f(x)$ (except perhaps if **needf** $= 0$).

7: **needg – int32 scalar**

8: **leng – int32 scalar**

9: **g(leng) – double array**

Concerns the calculations of the derivatives of the function $f(x)$.

leng is the length of the co-ordinate arrays **jgvar** and **igfun** in the call to e04vh.

needg indicates if **g** must be assigned during this call of **usrfun**.

needg $= 0$

> **g** is not required and is ignored.

needg > 0

> The partial derivatives of $f(x)$ must be calculated and assigned to **g**. For each $k = 1 : $ **leng**, the value of **g**(k) should be $\dfrac{\partial f_i(x)}{\partial x_j}$, where $i =$ **igfun**(k), $j =$ **jgvar**(k), for $k = 1, 2, \ldots,$ **leng**.

g contains the computed derivatives $G(x)$ (except perhaps if **needg** $= 0$).

These derivative elements must be stored in **g** in exactly the same positions as implied by the definitions of arrays **igfun** and **jgvar**. There is no internal check for consistency (except indirectly via the optional parameter **Verify Level**), so great care is essential.

10: **user – Any MATLAB object**

usrfun is called from e04vh with **user** as supplied to e04vh

Output Parameters

1: **status – int32 scalar**

Indicates the first and last calls to **usrfun**.

status $= 0$

There is nothing special about the current call to **usrfun**.

status $= 1$

e04vh is calling your (sub)program for the *first* time. You may wish to do something special such as read data from a file.

status ≥ 2

e04vh is calling your (sub)program for the *last* time. This parameter setting allows you to perform some additional computation on the final solution.

status $= 2$

The current **x** is *optimal*.

status $= 3$

The problem appears to be infeasible.

status $= 4$

The problem appears to be unbounded.

status $= 5$

An iterations limit was reached.

If the functions are expensive to evaluate, it may be desirable to do nothing on the last call. The first executable statement could be

```
if (status ≥ 2)
  return;
end
```

May be used to indicate that you are unable to evaluate f or its gradients at the current x. (For example, the problem functions may not be defined there.)

During the linesearch, $f(x)$ is evaluated at points $x = x_k + \alpha p_k$ for various step lengths α, where $f(x_k)$ has already been evaluated satisfactorily. For any such x, if you set **status** $= -1$, e04vh will reduce α and evaluate f again (closer to x_k, where $f(x_k)$ is more likely to be defined).

If for some reason you wish to terminate the current problem, set **status** ≤ -2.

2: **f(nf) – double array**

Concerns the calculation of $f(x)$.

nf is the length of the full vector $F(x) = f(x) + Ax$ as defined in the call to e04vh.

needf indicates if **f** must be assigned during this call of **usrfun**.

needf $= 0$

f is not required and is ignored.

needf > 0

The components of $f(x)$ corresponding to the nonlinear part of $F(x)$ must be calculated and assigned to **f**.

If $F_i(x)$ is linear and completely defined by the ith row of A, A_i', then the associated value $f_i(x)$ is ignored and need not be assigned. However, if $F_i(x)$ has a nonlinear portion $f_i(x)$ that happens to be zero at x, then it is still necessary to set $f_i(x) = 0$.

If the linear part A_i' of a nonlinear $F_i(x)$ is provided using the arrays **iafun**, **javar** and **a**, then it must not be computed again as part of $f_i(x)$.

To simplify the code, you may ignore the value of **needf** and compute $f(x)$ on every entry to **usrfun**.

needf may also be ignored with **Derivative Linesearch** and **Derivative Option** $= 1$. In this case, **needf** is always 1, and **f** must always be assigned.

f contains the computed functions $f(x)$ (except perhaps if **needf** $= 0$).

3: **g**(**leng**) – **double array**

Concerns the calculations of the derivatives of the function $f(x)$.

leng is the length of the co-ordinate arrays **jgvar** and **igfun** in the call to e04vh.

needg indicates if **g** must be assigned during this call of **usrfun**.

needg $= 0$

g is not required and is ignored.

needg > 0

The partial derivatives of $f(x)$ must be calculated and assigned to **g**. For each $k = 1 :$ **leng**, the value of **g**(k) should be $\frac{\partial f_i(x)}{\partial x_j}$, where $i =$ **igfun**(k), $j =$ **jgvar**(k), for $k = 1, 2, \ldots,$ **leng**.

g contains the computed derivatives $G(x)$ (except perhaps if **needg** $= 0$).

These derivative elements must be stored in **g** in exactly the same positions as implied by the definitions of arrays **igfun** and **jgvar**. There is no internal check for consistency (except indirectly via the optional parameter **Verify Level**), so great care is essential.

4: **user** – **Any MATLAB object**

usrfun is called from e04vh with **user** as supplied to e04vh

6: **iafun**(**lena**) – **int32 array**
7: **javar**(**lena**) – **int32 array**

8: **nea** – **int32 scalar**

Is the number of nonzero entries in A such that $F(x) = f(x) + Ax$.

Constraint: $0 \leq$ **nea** $\leq$ **lena**.

9: **a**(**lena**) – **double array**

Define the co-ordinates (i,j) and values A_{ij} of the nonzero elements of the linear part A of the function $F(x) = f(x) + Ax$.

In particular, **nea** triples (**iafun**(k), **javar**(k), **a**(k)) define the row and column indices $i =$ **iafun**(k) and $j =$ **javar**(k) of the element $A_{ij} =$ **a**(k).

The co-ordinates may define the elements of A in any order.

10: **igfun**(**leng**) – **int32 array**
11: **jgvar**(**leng**) – **int32 array**

Define the co-ordinates (i,j) of the nonzero elements of G, the nonlinear part of the derivative $J(x) = G(x) + A$ of the function $F(x) = f(x) + Ax$. e04vj may be used to define these two arrays.

The co-ordinates can define the elements of G in any order. However, user-supplied (sub)program **usrfun** must define the actual elements of **g** in exactly the same order as defined by the co-ordinates (**igfun**, **jgvar**).

12: **neg – int32 scalar**

The number of nonzero entries in G.

Constraint: $0 \leq$ **neg** $\leq$ **leng**.

13: **xlow(n) – double array**

14: **xupp(n) – double array**

Contain the lower and upper bounds l_x and u_x on the variables x.

To specify a nonexistent lower bound $[l_x]_j = -\infty$, set **xlow**$(j) \leq -infbnd$, where $infbnd$ is the optional parameter **Infinite Bound Size**. To specify a nonexistent upper bound $[u_x]_j = \infty$, set **xupp**$(j) \geq infbnd$.

To fix the jth variable at $x_j = \beta$, where $|\beta| < infbnd$, set **xlow**$(j) =$ **xupp**$(j) = \beta$.

Constraint: **xlow**$(i) \leq$ **xupp**(i), for $i = 1, 2, \ldots,$ **n**.

15: **xnames(nxname) – string array**

The optional names for the variables.

If **nxname** $= 1$, **xnames** is not referenced and default names will be used for output.

If **nxname** $=$ **n**, **xnames**(j) should contain the 8-character name of the jth variable.

16: **flow(nf) – double array**

17: **fupp(nf) – double array**

Contain the lower and upper bounds l_F and u_F on $F(x)$.

To specify a nonexistent lower bound $[l_F]_i = -\infty$, set **flow**$(i) \leq -infbnd$. For a nonexistent upper bound $[u_F]_i = \infty$, set **fupp**$(i) \geq infbnd$.

To make the ith constraint an *equality* at $F_i = \beta$, where $|\beta| < infbnd$, set **flow**$(i) =$ **fupp**$(i) = \beta$.

Constraint: **flow**$(i) \leq$ **fupp**(i), for $i = 1, 2, \ldots,$ **n**.

18: **fnames(nfname) – string array**

The optional names for the problem functions.

If **nfname** $= 1$, **fnames** is not referenced and default names will be used for output.

If **nfname** $=$ **nf**, **fnames**(i) should contain the 8-character name of the ith row of F.

19: **x(n) – double array**

An initial estimate of the variables x. See the following description of **xstate**.

20: **xstate(n) – int32 array**

The initial state for each variable x.

If **start** $= 0$ or 1 and no basis information is provided (the optional parameters **Old Basis File**, **Insert File** and **Load File** are all set to 0) **x** and **xstate** must be defined.

If nothing special is known about the problem, or if there is no wish to provide special information, you may set **x**$(j) = 0.0$, **xstate**$(j) = 0$, for all $j = 1, \ldots,$ **n**. If you set

x(j) = **xlow**(j) set **xstate**$(j) = 4$; if you set **x**(j) = **xupp**(j) then set **xstate**$(j) = 5$. In this case a Crash procedure is used to select an initial basis.

If **start** = 0 or 1 and basis information is provided (at least one of the optional parameters **Old Basis File**, **Insert File** and **Load File** is nonzero) **x** and **xstate** need not be set.

If **start** = 2 (Warm Start), **x** and **xstate** must be set (probably from a previous call). In this case **xstate**(j) must be 0, 1, 2 or 3, for $j = 1, \ldots,$ **n**.

Constraint: $0 \leq$ **xstate**$(j) \leq 5$, for $j = 1, \ldots,$ **n**.

21: **f(nf) – double array**

An initial value for the problem functions F. See the following description of **fstate**.

22: **fstate(nf) – int32 array**

The initial state for the problem functions F.

If **start** = 0 or 1 and no basis information is provided (the optional parameters **Old Basis File**, **Insert File** and **Load File** are all set to 0; the default, **f** and **fstate** must be defined.

If nothing special is known about the problem, or if there is no wish to provide special information, you may set **f**$(i) = 0.0$, **fstate**$(i) = 0$, for all $i = 1, \ldots,$ **nf**. Less trivially, to say that the optimal value of function **f**(i) will probably be equal to one of its bounds, set **f**(i) = **flow**(i) and **fstate**$(i) = 4$ or **f**(i) = **fupp**(i) and **fstate**$(i) = 5$ as appropriate. In this case a Crash procedure is used to select an initial basis.

If **start** = 0 or 1 and basis information is provided (at least one of the optional parameters **Old Basis File**, **Insert File** and **Load File** is nonzero), **f** and **fstate** need not be set.

If **start** = 2 (Warm Start), **f** and **fstate** must be set (probably from a previous call). In this case **fstate**(i) must be 0, 1, 2 or 3, for $i = 1, \ldots,$ **nf**.

Constraint: $0 \leq$ **fstate**$(i) \leq 5$, for $i = 1, 2, \ldots,$ **nf**.

23: **fmul(nf) – double array**

An estimate of γ, the vector of Lagrange multipliers (shadow prices) for the constraints $l_F \leq F(x) \leq u_F$. All **nf** components must be defined. If nothing is known about γ, set **fmul**$(i) = 0.0$, for $i = 1, 2, \ldots,$ **nf**. For warm start use the values from a previous call.

24: **ns – int32 scalar**

The number of superbasic variables. **ns** need not be specified for cold starts, but should retain its value from a previous call when warm start is used.

25: **cw(lencw) – string array**

Constraint: **lencw** ≥ 600.

26: **iw(leniw) – int32 array**

Constraint: **leniw** ≥ 600.

27: **rw(lenrw) – double array**

Constraint: **lenrw** ≥ 600.

5.2 Optional Input Parameters

1: **nf – int32 scalar**

Default: The dimension of the arrays **f**, **fstate**. (An error is raised if these dimensions are not equal.)

nf, the number of problem functions in $F(x)$, including the objective function (if any) and the linear and nonlinear constraints. Upper and lower bounds on x can be defined using the parameters **xlow** and **xupp** and should not be included in F.

Constraint: $\mathbf{nf} > 0$.

2: **n – int32 scalar**

Default: The dimension of the arrays **xlow**, **xupp**, **x**, **xstate**, **xmul**. (An error is raised if these dimensions are not equal.)

n, the number of variables.

Constraint: $\mathbf{n} > 0$.

3: **nxname – int32 scalar**

Default: The dimension of the array **xnames**.

the number of names provided in the array **xnames**.

nxname $= 1$

There are no names provided and generic names will be used in the output.

nxname $=$ **n**

Names for all variables must be provided and will be used in the output.

Constraint: **nxname** $= 1$ or **n**.

4: **nfname – int32 scalar**

Default: The dimension of the array **fnames**.

the number of names provided in the array **fnames**.

nfname $= 1$

There are no names provided and generic names will be used in the output.

nfname $=$ **nf**

Names for all functions must be provided and will be used in the output.

Constraint: **nfname** $= 1$ or **nf**.

Note: If **nxname** $= 1$ then **nfname** must also be 1 (and vice versa). Similarly, if **nxname** $=$ **n** then **nfname** must be **nf** (and vice versa).

5: **lena – int32 scalar**

Default: The dimension of the arrays **iafun**, **javar**, **a**. (An error is raised if these dimensions are not equal.)

Constraint: **lena** ≥ 1.

6: **leng – int32 scalar**

Default: The dimension of the arrays **igfun**, **jgvar**. (An error is raised if these dimensions are not equal.)

Constraint: **leng** ≥ 1.

7: **lencw – int32 scalar**

Default: The dimension of the array **cw**.

Constraint: **lencw** ≥ 600.

8: **leniw – int32 scalar**

Default: The dimension of the array **iw**.

Constraint: **leniw** ≥ 600.

9: **lenrw – int32 scalar**

Default: The dimension of the array **rw**.

Constraint: **lenrw** ≥ 600.

10: **user – Any MATLAB object**

user is not used by e04vh, but is passed to **usrfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **x(n) – double array**

The final values of the variable x.

2: **xstate(n) – int32 array**

The final state of the variables.

xstate(j)	**State of variable j**	**Usual value of x**(j)
0	nonbasic	**xlow**(j)
1	nonbasic	**xupp**(j)
2	superbasic	Between **xlow**(j) and **xupp**(j)
3	basic	Between **xlow**(j) and **xupp**(j)

Basic and superbasic variables may be outside their bounds by as much as the optional parameter **Minor Feasibility Tolerance**. Note that if scaling is specified, the feasibility tolerance applies to the variables of the *scaled* problem. In this case, the variables of the original problem may be as much as 0.1 outside their bounds, but this is unlikely unless the problem is very badly scaled. Check the value of `Primal infeasibility` output to the unit number associated with the optional parameter **Print File**.

Very occasionally some nonbasic variables may be outside their bounds by as much as the optional parameter **Minor Feasibility Tolerance**, and there may be some nonbasics for which **x**(j) lies strictly between its bounds.

If **ninf** > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by **sinf** if scaling was not used).

3: **xmul(n) – double array**

The vector of the dual variables (Lagrange multipliers) for the simple bounds $l_x \leq x \leq u_x$.

4: **f(nf) – double array**

The final values for the problem functions F (the values F at the final point **x**).

5: **fstate(nf) – int32 array**

The final state of the variables. The elements of **fstate** have the following meaning:

fstate(i)	**State of the corresponding slack variable**	**Usual value of f**(i)
0	nonbasic	**flow**(i)
1	nonbasic	**fupp**(i)
2	superbasic	Between **flow**(i) and **fupp**(i)
3	basic	Between **flow**(i) and **fupp**(i)

Basic and superbasic slack variables may lead to the corresponding functions being outside their bounds by as much as the optional parameter **Minor Feasibility Tolerance**.

Very occasionally some functions may be outside their bounds by as much as the optional parameter **Minor Feasibility Tolerance**, and there may be some nonbasics for which **f**(i) lies strictly between its bounds.

If **ninf** > 0, some basic and superbasic variables may be outside their bounds by an arbitrary amount (bounded by **sinf** if scaling was not used).

6: **fmul(nf) – double array**

The vector of the dual variables (Lagrange multipliers) for the general constraints $l_F \leq F(x) \leq u_F$

7: **ns – int32 scalar**

The final number of superbasic variables.

8: **ninf – int32 scalar**

9: **sinf – double scalar**

Are the number and the sum of the infeasibilities of constraints that lie outside one of their bounds by more than the optional parameter **Minor Feasibility Tolerance** *before the solution is unscaled*.

If any *linear* constraints are infeasible, x minimizes the sum of the infeasibilities of the linear constraints subject to the upper and lower bounds being satisfied. In this case **ninf** gives the number of variables and linear constraints lying outside their upper or lower bounds. The nonlinear constraints are not evaluated.

Otherwise, x minimizes the sum of infeasibilities of the *nonlinear* constraints subject to the linear constraints and upper and lower bounds being satisfied. In this case **ninf** gives the number of components of $F(x)$ lying outside their bounds by more than the optional parameter **Minor Feasibility Tolerance**. Again this is *before the solution is unscaled.*

10: **cw(lencw) – string array**

11: **iw(leniw) – int32 array**

12: **rw(lenrw) – double array**

13: **user – Any MATLAB object**

user is not used by e04vh, but is passed to **usrfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

14: **ifail – int32 scalar**

0 unless the routine detects an error (see Section 6).

6 Error Indicators and Warnings

Note: e04vh may return useful information for one or more of the following detected errors or warnings.

ifail $= 1$

The initialization routine e04vg has not been called or at least one of **lencw**, **leniw** and **lenrw** is less than 600.

ifail $= 2$

An input parameter is invalid. The output message provides more details of the invalid argument.

ifail $= 3$

Requested accuracy could not be achieved.

A feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but e04vh is within 10^{-2} of satisfying the **Major Optimality Tolerance**. Check that the **Major Optimality Tolerance** is not too small.

ifail $= 4$

The problem appears to be infeasible.

When the constraints are linear, this message is based on a relatively reliable indicator of infeasibility. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax - s = 0$ (see (6) and (7) in Section 10.2), there is apparently no point that satisfies the bounds on x and s. Violations as small as the **Minor Feasibility Tolerance** are ignored, but at least one component of x or s violates a bound by more than the tolerance.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, e04vh is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), e04vh enters so-called ‘nonlinear elastic’ mode. The subproblem includes the original QP objective and the sum of the infeasibilities – suitably weighted using the optional parameter **Elastic Weight**. In elastic mode, some of the bounds on the nonlinear rows are ‘elastic’ – i.e., they are allowed to violate their specific bounds. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, e04vh will tend to determine a ‘good’ infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, e04vh would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though e04vh locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

ifail $= 5$

The problem appears to be unbounded (or badly scaled).

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. Consider adding an upper or lower bound to the variable. Also, examine the constraints that have nonzeros in the associated column, to see if they have been formulated as intended.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the optional parameter **Scale Option**.

For nonlinear problems, e04vh monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the unbounded parameters (see Section)), the problem is terminated and declared unbounded. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The message may indicate an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the **Violation Limit**.

ifail = 6

Iteration limit reached.

Either the **Iterations Limit** or the **Major Iterations Limit** was exceeded before the required solution could be found. Check the iteration log to be sure that progress was being made. If so, restart the run using a Basis file that was saved (or should have been saved) at the end of the run.

If none of the above limits have been reached, this error may mean that the problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a pricing operation (where a nonbasic variable is selected to become superbasic) is necessary to continue, but it can't continue as the number of superbasic variables has already reached the limit specified by the optional parameter **Superbasics Limit**. In general, raise the **Superbasics Limit** s by a reasonable amount, bearing in mind the storage needed for the reduced Hessian.

ifail = 7

Numerical difficulties have been encountered and no further progress can be made.

Several circumstances could lead to this exit.

1. The user-supplied (sub)program **usrfun** could be returning accurate function values but inaccurate gradients (or vice versa). This is the most likely cause. Study the comments given for **ifail** = 8, and do your best to ensure that the coding is correct.
2. The function and gradient values could be consistent, but their precision could be too low. For example, accidental use of a ***real*** data type when double was intended would lead to a relative function precision of about 10^{-6} instead of something like 10^{-15}. The default **Major Optimality Tolerance** of 2×10^{-6} would need to be raised to about 10^{-3} for optimality to be declared (at a rather suboptimal point). Of course, it is better to revise the function coding to obtain as much precision as economically possible.
3. If function values are obtained from an expensive iterative process, they may be accurate to rather few significant figures, and gradients will probably not be available. One should specify

   Function Precision t

   Major Optimality Tolerance $\sqrt{t}$

   but even then, if t is as large as 10^{-5} or 10^{-6} (only 5 or 6 significant figures), the same exit condition may occur. At present the only remedy is to increase the accuracy of the function calculation.
4. An LU factorization of the basis has just been obtained and used to recompute the basic variables x_B, given the present values of the superbasic and nonbasic variables. A step of 'iterative refinement' has also been applied to increase the accuracy of x_B. However, a row check has revealed that the resulting solution does not satisfy the current constraints $Ax - s = 0$ sufficiently well.

   This probably means that the current basis is very ill-conditioned. If there are some linear constraints and variables, try **Scale Option** = 1 if scaling has not yet been used.

For certain highly structured basis matrices (notably those with band structure), a systematic growth may occur in the factor U. Consult the description of `Umax` and `Growth` in Section 12.4 and set the **LU Factor Tolerance** to 2.0 (or possibly even smaller, but not less than 1.0).

5. The first factorization attempt will have found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix U were respectively zero or smaller than a certain tolerance.) The associated variables are replaced by slacks and the modified basis is refactorized, but singularity persists. This must mean that the problem is badly scaled, or the **LU Factor Tolerance** is too much larger than 1.0. This is highly unlikely to occur.

ifail $= 8$

Derivative appears to be incorrect.

A check has been made on some elements of the Jacobian as returned in the parameter **g** of user-supplied (sub)program **usrfun**. At least one value disagrees remarkably with its associated forward difference estimate (the relative difference between the computed and estimated values is 1.0 or more). This exit is a safeguard, since e04vh will usually fail to make progress when the computed gradients are seriously inaccurate. In the process it may expend considerable effort before terminating with **ifail** $= 7$.

Check the function and Jacobian computation *very carefully* in user-supplied (sub)program **usrfun**. A simple omission could explain everything. If a component is very large, then give serious thought to scaling the function or the nonlinear variables.

If you feel *certain* that the computed Jacobian is correct (and that the forward-difference estimate is therefore wrong), you can specify **Verify Level** $= 0$ to prevent individual elements from being checked. However, the optimization procedure may have difficulty.

ifail $= 9$

Undefined user-supplied function.

You have indicated that the problem functions are undefined by assigning the value **status** $= -1$ on exit from **usrfun**. e04vh attempts to evaluate the problem functions closer to a point at which the functions are already known to be defined. This exit occurs if e04vh is unable to find a point at which the functions are defined. This will occur in the case of:

– undefined functions with no recovery possible;

– undefined functions at the first point;

– undefined functions at the first feasible point; or

– undefined functions when checking derivatives.

ifail $= 10$

User requested termination.

You have indicated the wish to terminate solution of the current problem by setting **status** to a value < -1 on exit from **usrfun**.

ifail $= 11$

Internal memory allocation failed when attempting to obtain the required workspace. Please contact NAG.

ifail $= 12$

Internal memory allocation was insufficient. Please contact NAG.

ifail $= 13$

An error has occurred in the basis package, perhaps indicating incorrect setup of arrays. Set the optional parameter **Print File** and examine the output carefully for further information.

ifail $= 14$

An unexpected error has occurred. Set the optional parameter **Print File** and examine the output carefully for further information.

7 Accuracy

If the value of the optional parameter **Major Optimality Tolerance** is set to 10^{-d} (default value $= \sqrt{\epsilon}$) and **ifail** $= 0$ on exit, then the final value of $f(x)$ should have approximately d correct significant digits.

8 Further Comments

This section describes the final output produced by e04vh. Intermediate and other output are given in Section 12.

8.1 The Final Output

If **Print File** > 0, the final output, including a listing of the status of every variable and constraint will be sent to the channel numbers associated with **Print File**. The following describes the output for each constraint (row) and variable (column).

8.1.1 The ROWS section

General linear constraints take the form $l \leq A_L x \leq u$. The ith constraint is therefore of the form

$$\alpha \leq \nu_i x \leq \beta,$$

where ν_i is the ith row of A_L.

Internally, the constraints take the form $A_L x - s = 0$, where s is the set of slack variables (which satisfy the bounds $l \leq s \leq u$). For the ith row it is the slack variable s_i that is directly available and it is sometimes convenient to refer to its state. Nonlinear constraints $\alpha \leq f_i(x) + \nu_i x \leq \beta$ are treated similarly, except that the row activity and degree of infeasibility are computed directly from $f_i(x) + \nu_i x$, rather than s_i.

A '.' is printed for any numerical value that is exactly zero.

Label	**Description**
`Number`	is the value of $n + i$. (This is used internally to refer to s_i in the intermediate output.)
`Row`	gives the name of the ith row.
`State`	the state of the ith row relative to the bounds α and β. The various states possible are as follows:

`LL`	the row is at its lower limit, α.
`UL`	the row is at its upper limit, β.
`EQ`	the limits are the same ($\alpha = \beta$).
`FR`	s_i is nonbasic and currently zero, even though it is free to take any value between its bounds α and β.
`BS`	s_i is basic.
`SBS`	s_i is superbasic.

A key is sometimes printed before `State`. Note that unless the optional parameter **Scale Option** $= 0$ is specified, the tests for assigning a key are applied to the variables of the scaled problem.

`A`	*Alternative optimum possible.* The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the other free variables *might* change,

	giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers *might* also change.
`D`	*Degenerate.* The variable is basic or superbasic, but it is equal (or very close) to one of its bounds.
`I`	*Infeasible.* The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the **Feasibility Tolerance**.
`N`	*Not precisely optimal.* The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Major Optimality Tolerance**, the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

`Activity`	is the value of $\nu_i x$ (or $f_i(x) + \nu_i x$ for nonlinear rows) at the final iterate.
`Slack Activity`	is the value by which the row differs from its nearest bound. (For the free row (if any), it is set to `Activity`.)
`Lower Limit`	is α, the lower bound on the row.
`Upper Limit`	is β, the upper bound on the row.
`Dual Activity`	is the value of the dual variable π_i (the Lagrange multiplier for the ith constraint). The full vector π always satisfies $B^{\mathrm{T}}\pi = g_B$, where B is the current basis matrix and g_B contains the associated gradients for the current objective function. For FP problems, π_i is set to zero.
`i`	gives the index i of the ith row.

8.1.2 The COLUMNS section

Let the jth component of x be the variable x_j and assume that it satisfies the bounds $\alpha \leq x_j \leq \beta$. A '.' is printed for any numerical value that is exactly zero.

Label	**Description**
`Number`	is the column number j. (This is used internally to refer to x_j in the intermediate output.)
`Column`	gives the name of x_j.
`State`	the state of x_j relative to the bounds α and β. The various states possible are as follows:

`LL`	x_j is nonbasic at its lower limit, α.
`UL`	x_j is nonbasic at its upper limit, β.
`EQ`	x_j is nonbasic and fixed at the value $\alpha = \beta$.
`FR`	x_j is nonbasic at some value strictly between its bounds: $\alpha < x_j < \beta$.
`BS`	x_j is basic. Usually $\alpha < x_j < \beta$.
`SBS`	x_j is superbasic. Usually $\alpha < x_j < \beta$.

A key is sometimes printed before `State`. Note that unless the optional parameter **Scale Option** $= 0$ is specified, the tests for assigning a key are applied to the variables of the scaled problem.

`A`	*Alternative optimum possible.* The variable is nonbasic, but its reduced gradient is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change in the value of the objective function. The values of the other free variables *might* change,

giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case, the values of the Lagrange multipliers *might* also change.

`D`	*Degenerate.* The variable is basic or superbasic, but it is equal (or very close) to one of its bounds.
`I`	*Infeasible.* The variable is basic or superbasic and is currently violating one of its bounds by more than the value of the **Feasibility Tolerance**.
`N`	*Not precisely optimal.* The variable is nonbasic or superbasic. If the value of the reduced gradient for the variable exceeds the value of the optional parameter **Major Optimality Tolerance**, the solution would not be declared optimal because the reduced gradient for the variable would not be considered negligible.

`Activity`	is the value of x_j at the final iterate.
`Obj Gradient`	is the value of g_j at the final iterate. For FP problems, g_j is set to zero.
`Lower Limit`	is the lower bound specified for the variable. `None` indicates that **xlow**$(j) \leq -infbnd$.
`Upper Limit`	is the upper bound specified for the variable. `None` indicates that **xupp**$(j) \geq infbnd$.
`Reduced Gradnt`	is the value of the reduced gradient $d_j = g_j - \pi^{\mathrm{T}} a_j$ where a_j is the jth column of the constraint matrix. For FP problems, d_j is set to zero.
`m + j`	is the value of $m + j$.

Note that movement off a constraint (as opposed to a variable moving away from its bound) can be interpreted as allowing the entry in the `Slack Activity` column to become positive.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.

9 Example

e04vh_usrfun.m

```
function [status, f, g, user] = usrfun(status, n, x, needf, nf, f,
needg, leng, g, user)

  if (needf > 0)
%   The nonlinear components of f_i(x) need to be assigned,
%   for i = 1 to NF
    f(1) = 1000*sin(-x(1)-0.25) + 1000*sin(-x(2) -0.25);
    f(2) = 1000*sin(x(1)-0.25) + 1000*sin(x(1)-x(2) -0.25);
    f(3) = 1000*sin(x(2)-x(1)-0.25) + 1000*sin(x(2) -0.25);
%   n.b. in this example there is no need to assign for the wholly
%   linear components f_4(x) and f_5(x).
    f(6) = 1.0d-6*x(3)^3 + 2.0d-6*x(4)^3/3;
  end

  if (needg > 0)
%   The derivatives of the function f_i(x) need to be assigned.
%   G(k) should be set to partial derivative df_i(x)/dx_j where
%   i = IGFUN(k) and j = IGVAR(k), for k = 1 to LENG.
    g(1) = -1000*cos(-x(1)-0.25);
    g(2) = -1000*cos(-x(2)-0.25);
    g(3) = 1000*cos(x(1)-0.25) + 1000*cos(x(1)-x(2) -0.25);
    g(4) = -1000*cos(x(1)-x(2)-0.25);
    g(5) = -1000*cos(x(2)-x(1)-0.25);
    g(6) = 1000*cos(x(2)-x(1)-0.25) + 1000*cos(x(2) -0.25);
    g(7) = 3.0d-6*x(3)^2;
    g(8) = 2.0d-6*x(4)^2;
  end
```

```
start = int32(0);
objadd = 0;
objrow = int32(6);
prob = '        ';
iafun = [int32(1);
     int32(2);
     int32(4);
     int32(4);
     int32(5);
     int32(5);
     int32(6);
     int32(6)];
javar = [int32(3);
     int32(4);
     int32(1);
     int32(2);
     int32(1);
     int32(2);
     int32(3);
     int32(4)];
nea = int32(8);
a = [-1;
     -1;
     -1;
     1;
     1;
     -1;
     3;
     2];
igfun = [int32(1);
     int32(1);
     int32(2);
     int32(2);
     int32(3);
     int32(3);
     int32(6);
     int32(6)];
jgvar = [int32(1);
     int32(2);
     int32(1);
     int32(2);
     int32(1);
     int32(2);
     int32(3);
     int32(4)];
neg = int32(8);
xlow = [-0.55;
     -0.55;
     0;
     0];
xupp = [0.55;
     0.55;
     1200;
     1200];
xnames = {'X1       '; 'X2       '; 'X3       '; 'X4       '};
flow = [-894.8;
     -894.8;
     -1294.8;
     -0.55;
     -0.55;
     -9.999999999999999e+24];
fupp = [-894.8;
     -894.8;
     -1294.8;
     9.999999999999999e+24;
     9.999999999999999e+24;
     9.999999999999999e+24];
fnames = {'NlnCon 1'; 'NlnCon 2'; 'NlnCon 3'; 'LinCon 1'; 'LinCon 2';
'Objectiv'};
x = [0;
     0;
     0;
     0];
xstate = zeros(4, 1, 'int32');
f = [0;
     0;
     0;
     0;
     0;
     0];
fstate = zeros(6, 1, 'int32');
fmul = [0;
     0;
     0;
     0;
     0;
     0];
ns = int32(0);
[cw, iw, rw, ifail] = e04vg();
[xOut, xstateOut, xmul, fOut, fstateOut, fmulOut, nsOut, ninf, sinf, ...
 cwOut, iwOut, rwOut, cuser, ifail] = ...
     e04vh(start, objadd, objrow, prob, 'e04vh_usrfun', iafun, javar, ...
              nea, a, igfun, jgvar, neg, xlow, xupp, xnames, flow, fupp,
...
            fnames, x, xstate, f, fstate, fmul, ns, cw, iw, rw);
```

Note: *the remainder of this document is intended for more advanced users. Section 10 contains a detailed description of the algorithm which may be needed in order to understand Sections 11 and 12. Section 11 describes the optional parameters which may be set by calls to e04vl, e04vm and/or e04vn. Section 12 describes the quantities which can be requested to monitor the course of the computation.*

10 Algorithmic Details

Here we summarize the main features of the SQP algorithm used in e04vh and introduce some terminology used in the description of the (sub)program and its arguments. The SQP algorithm is fully described in Gill *et al.* (2002).

10.1 Constraints and Slack Variables

Problem (1) contains n variables in x. Let m be the number of components of $f(x)$ and A_Lx combined. The upper and lower bounds on those terms define the *general constraints* of the problem. e04vh converts the general constraints to equalities by introducing a set of *slack variables* $s = (s_1, s_2, \dots, s_m)^{\mathrm{T}}$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq \infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq \infty$. The minimization problem (1) can therefore be written in the equivalent form

$$\underset{x,s}{\text{minimize}} f_0(x) \qquad \text{subject to} \begin{pmatrix} f(x) \\ A_Lx \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. \tag{3}$$

The general constraints become the equalities $f(x) - s_N = 0$ and $A_Lx - s_L = 0$, where s_L and s_N are the *linear* and *nonlinear* slacks.

10.2 Major Iterations

The basic structure of the SQP algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that satisfy the linear constraints and converge to a point that satisfies the nonlinear constraints and the first-order conditions for optimality. At each iterate x_k a QP subproblem is used to generate a search direction towards the next iterate x_{k+1}. The constraints of the subproblem are formed from the linear constraints $A_L x - s_L = 0$ and the linearized constraint

$$f(x_k) + f'(x_k)(x - x_k) - s_N = 0, \tag{4}$$

where $f'(x_k)$ denotes the *Jacobian matrix*, whose elements are the first derivatives of $f(x)$ evaluated at x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{array}{rcl} f'(x_k)x - s_N & = & -f(x_k) + f'(x_k)x_k, \\ A_L x \quad - s_L & = & 0, \end{array} \tag{5}$$

where x and s are bounded above and below by u and l as before. If the m by n matrix A and m-vector b are defined as

$$A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix}, \tag{6}$$

then the QP subproblem can be written as

$$\underset{x,s}{\text{minimize}}\, q(x, x_k) = g_k^{\mathrm{T}}(x - x_k) + \frac{1}{2}(x - x_k)^{\mathrm{T}} H_k (x - x_k) \quad \text{subject to } Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \tag{7}$$

where $q(x, x_k)$ is a quadratic approximation to a modified Lagrangian function (see Gill *et al.* (2002)). The matrix H_k is a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration. If some of the variables enter the Lagrangian linearly the Hessian will have some zero rows and columns. If the nonlinear variables appear first, then only the n_1 rows and columns of the Hessian need to be approximated, where n_1 is the number of nonlinear variables. This quantity is determined by the implicit values of the number of nonlinear objective and Jacobian variables determined after the constraints and variables are reordered.

10.3 Minor Iterations

Solving the QP subproblem is itself an iterative procedure. Here, the iterations of the QP solver e04nq form the *minor* iterations of the SQP method. e04nq uses a reduced-Hessian active-set method implemented as a reduced-gradient method. At each minor iteration, the constraints $Ax - s = b$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = b, \tag{8}$$

where the *basis matrix* B is square and nonsingular, and the matrices S and N are the remaining columns of $(A \quad -I)$. The vectors x_B, x_S and x_N are the associated *basic*, *superbasic* and *nonbasic* variables respectively; they are a permutation of the elements of x and s. At a QP subproblem, the basic and superbasic variables will lie somewhere between their bounds, while the nonbasic variables will normally be equal to one of their bounds. At each iteration, x_S is regarded as a set of independent variables that are free to move in any desired direction, namely one that will improve the value of the QP objective (or the sum of infeasibilities). The basic variables are then adjusted in order to ensure that (x, s) continues to satisfy $Ax - s = b$. The number of superbasic variables (n_S, say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of *how nonlinear* the problem is. In particular, n_S will always be zero for LP problems.

If it appears that no improvement can be made with the current definition of B, S and N, a nonbasic variable is selected to be added to S, and the process is repeated with the value of n_S increased by one. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of n_S is decreased by one.

Associated with each of the m equality constraints $Ax - s = b$ are the *dual variables* π. Similarly, each variable in (x, s) has an associated *reduced gradient* d_j. The reduced gradients for the variables x are the quantities $g - A^{\mathrm{T}}\pi$, where g is the gradient of the QP objective, and the reduced gradients for the slacks are the dual variables π. The QP subproblem is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for other variables, including superbasics. In practice, an *approximate* QP solution $(\hat{x}_k, \hat{s}_k, \hat{\pi}_k)$ is found by relaxing these conditions.

10.4 The Merit Function

After a QP subproblem has been solved, new estimates of the solution are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f_0(x) - \pi^{\mathrm{T}}(f(x) - s_N) + \frac{1}{2}(f(x) - s_N)^{\mathrm{T}}D(f(x) - s_N), \tag{9}$$

where D is a diagonal matrix of penalty parameters $(D_{ii} \geq 0)$, and π now refers to dual variables for the nonlinear constraints in (1). If (x_k, s_k, π_k) denotes the current solution estimate and $(\hat{x}_k, \hat{s}_k, \hat{\pi}_k)$ denotes the QP solution, the linesearch determines a step α_k $(0 < \alpha_k \leq 1)$ such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \hat{x}_k - x_k \\ \hat{s}_k - s_k \\ \hat{\pi}_k - \pi_k \end{pmatrix} \tag{10}$$

gives a *sufficient decrease* in the merit function $\mathcal{M}$. When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ (see Gill *et al.* (1992)). The value of s_N is adjusted to minimize the merit function as a function of s before the solution of the QP subproblem (see Gill *et al.* (1986c) and Eldersveld (1991)).

10.5 Treatment of Constraint Infeasibilities

e04vh makes explicit allowance for infeasible constraints. First, infeasible *linear* constraints are detected by solving the linear program

$$\underset{x,v,w}{\text{minimize}}\, e^{\mathrm{T}}(v + w) \qquad \text{subject to } l \leq \begin{pmatrix} x \\ A_L x - v + w \end{pmatrix} \leq u, \qquad v \geq 0, \qquad w \geq 0, \tag{11}$$

where e is a vector of ones, and the nonlinear constraint bounds are temporarily excluded from l and u. This is equivalent to minimizing the sum of the general linear constraint violations subject to the bounds on x. (The sum is the ℓ_1-norm of the linear constraint violations. In the linear programming literature, the approach is called *elastic programming*.)

The linear constraints are infeasible if the optimal solution of (11) has $v \neq 0$ or $w \neq 0$. e04vh then terminates without computing the nonlinear functions.

Otherwise, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) e04vh proceeds to solve nonlinear problems as given, using search directions obtained from the sequence of QP subproblems (see (7)).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), e04vh enters 'elastic' mode and thereafter solves the problem

$$\underset{x,v,w}{\text{minimize}} f_0(x) + \gamma e^{\mathrm{T}}(v + w) \qquad \text{subject to } l \leq \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \leq u, \qquad v \geq 0, \qquad w \geq 0, \tag{12}$$

where γ is a nonnegative parameter (the *elastic weight*), and $f_0(x) + \gamma e^{\mathrm{T}}(v + w)$ is called a *composite objective* (the ℓ_1 penalty function for the nonlinear constraints).

The value of γ may increase automatically by multiples of 10 if the optimal v and w continue to be nonzero. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds.

The initial value of γ is controlled by the optional parameter **Elastic Weight**.

11 Optional Parameters

Several optional parameters in e04vh define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of e04vh these optional parameters have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters. A complete list of optional parameters and their default values is given in Section 11.1.

Optional parameters may be specified by calling one, or more, of the routines e04vl, e04vm and e04vn before a call to e04vh.

e04vl, e04vm and e04vn can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
[cw, iw, rw, ifail] = e04vl('Print Level = 5', cw, iw, rw);
```

e04vl, e04vm and e04vn should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by you are set to their default values. Optional parameters specified by you are unaltered by e04vh (unless they define invalid values) and so remain in effect for subsequent calls to e04vh, unless altered by you.

11.1 Optional Parameter Checklist and Default Values

The following list gives the valid options. For each option, we give the keyword, any essential optional qualifiers and the default value. A definition for each option can be found in Section 11.2. The minimum abbreviation of each keyword is underlined. If no characters of an optional qualifier are underlined, the qualifier may be omitted. The letter a denotes a phrase (character string) that qualifies an option. The letters i and r denote integer and double values required with certain options. The number ϵ is a generic notation for ***machine precision*** (see x02aj), and ϵ_r denotes the relative precision of the objective function (see optional parameter **Function Precision**). The variable $infbnd$ holds the value of **Infinite Bound Size**.

Optional Parameters	**Default Values**
Backup Basis File	Default $= 0$. See **New Basis File**.
Central Difference Interval	Default $= \epsilon_r^{\frac{1}{3}}$
Check Frequency	Default $= 60$
Crash Option	Default $= 3$
Crash Tolerance	Default $= 0.1$. See **Crash Option**.
Defaults	
Derivative Linesearch	Default
Derivative Option	Default $= 1$
Difference Interval	Default $= \sqrt{\epsilon_r}$
Dump File	Default $= 0$
Elastic Weight	Default $= 10^4$
Expand Frequency	Default $= 10000$
Factorization Frequency	Default $= 50$
Feasibility Tolerance	Default $= \max\{10^{-6}, \sqrt{\epsilon}\}$. See **Minor Feasibility Tolerance**.
Feasible Point	See **Minimize**
Function Precision	Default $= \epsilon^{0.8}$
Hessian Frequency	Default $= 99999999$
Hessian Full Memory	Default if $n_1 \leq 75$
Hessian Limited Memory	Default if $n_1 > 75$. See **Hessian Full Memory**.
Hessian Updates	Default $=$ **Hessian Frequency** if **Hessian Full Memory**, 10 otherwise

Infinite Bound Size	Default $= 10^{20}$
Insert File	Default $= 0$. See **Punch File**.
Iterations Limit	Default $= \max(10000, 10\max(\mathbf{n}, \mathbf{nf}))$
LU Complete Pivoting	See **LU Partial Pivoting**
LU Density Tolerance	Default $= 0.6$
LU Factor Tolerance	Default $= 3.99$
LU Partial Pivoting	Default
LU Rook Pivoting	See **LU Partial Pivoting**
LU Singularity Tolerance	Default $= \epsilon^{\frac{2}{3}}$. See **LU Density Tolerance**.
LU Update Tolerance	Default $= 3.99$. See **LU Factor Tolerance**.
Linesearch Tolerance	Default $= 0.9$
List	See **Nolist**
Load File	Default $= 0$. See **Dump File**.
Major Feasibility Tolerance	Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$
Major Iterations Limit	Default $= \max(1000, 3\max(n, nf))$
Major Optimality Tolerance	Default $= 2\max\left(10^{-6}, \sqrt{\epsilon}\right)$
Major Print Level	Default $= 000001$
Major Step Limit	Default $= 2.0$
Maximize	See **Minimize**
Minimize	Default
Minor Feasibility Tolerance	Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$
Minor Iterations Limit	Default $= 500$
Minor Print Level	Default $= 1$
New Basis File	Default $= 0$
New Superbasics Limit	Default $= 99$
Nolist	Default
Nonderivative Linesearch	See **Derivative Linesearch**
Old Basis File	Default $= 0$
Partial Price	Default $= 1$
Pivot Tolerance	Default $= \epsilon^{\frac{2}{3}}$
Print File	Default $= 0$
Print Frequency	Default $= 100$
Proximal Point Method	Default $= 1$
Punch File	Default $= 0$
Save Frequency	Default $= 100$. See **New Basis File**.
Scale Option	Default $= 0$
Scale Print	See **Scale Option**
Scale Tolerance	Default $= 0.9$. See **Scale Option**.
Solution File	Default $= 0$
Summary File	Default $= 0$
Summary Frequency	Default $= 100$. See **Summary File**.
Superbasics Limit	Default $= n_1$
Suppress Parameters	
System Information No	Default
System Information Yes	See **System Information No**
Timing Level	Default $= 0$
Unbounded Objective	Default $=$ 1.0D+15
Unbounded Step Size	Default $= infbnd$. See **Unbounded Objective**.
Verify Level	Default $= 0$
Violation Limit	Default $=$ 1.0D+6

11.2 Description of the Optional Parameters

Central Difference Interval r Default $= \epsilon_r^{\frac{1}{3}}$

When **Derivative Option** $= 0$, the central-difference interval r is used near an optimal solution to obtain more accurate (but more expensive) estimates of gradients. Twice as many function evaluations are required compared to forward differencing. The interval used for the jth variable is

$h_j = r(1 + |x_j|)$. The resulting derivative estimates should be accurate to $O(r^2)$, unless the functions are badly scaled.

Check Frequency i Default $= 60$

Every ith minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (the linear constraints and the linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately. If $i \leq 0$, the value of $i = 99999999$ is used and effectively no checks are made.

Check Frequency $= 1$ is useful for debugging purposes, but otherwise this option should not be needed.

Crash Option i Default $= 3$
Crash Tolerance r Default $= 0.1$

Except on restarts, an internal Crash procedure is used to select an initial basis from certain rows and columns of the constraint matrix $(A \quad -I)$. The **Crash Option** i determines which rows and columns of A are eligible initially, and how many times the Crash procedure is called. Columns of $-I$ are used to pad the basis where necessary.

i	Meaning
0	The initial basis contains only slack variables: $B = I$.
1	The Crash procedure is called once, looking for a triangular basis in all rows and columns of A.
2	The Crash procedure is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and the Crash procedure is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).
3	The Crash procedure is called up to three times (if there are nonlinear constraints). The first two calls treat *linear equalities* and *linear inequalities* separately. As before, the last call treats nonlinear rows before the first major iteration.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to a bound). The Crash procedure then makes several passes through the columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to 'pivot' on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For remaining unassigned rows, slack variables are inserted to complete the basis.

The **Crash Tolerance** r allows the starting Crash procedure to ignore certain 'small' nonzeros in each column of A. If $a_{\max}$ is the largest element in column j, other nonzeros of a_{ij} in the columns are ignored if $|a_{ij}| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)

When $r > 0.0$, the basis obtained by the Crash procedure may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A form the matrix shown under **LU Factor Tolerance**; i.e., a tridiagonal matrix with entries -1, 4, -1. To help the Crash procedure choose all m columns for the initial basis, we would specify a **Crash Tolerance** of r for some value of $r > 0.5$.

Defaults

This special keyword may be used to reset all optional parameters to their default values.

Derivative Option i Default $= 1$

Optional parameter **Derivative Option** specifies which nonlinear function gradients are known analytically and will be supplied to e04vh by user-supplied (sub)program **usrfun**.

i	**Meaning**
0	Some problem derivatives are unknown.
1	All problem derivatives are known.

The value $i = 1$ should be used whenever possible. It is the most reliable and will usually be the most efficient.

If $i = 0$, e04vh will *estimate* the missing components of $G(x)$ using finite differences. This may simplify the coding of user-supplied (sub)program **usrfun**. However, it could increase the total run-time substantially (since a special call to **usrfun** is required for each column of the Jacobian that has a missing element), and there is less assurance that an acceptable solution will be located. If the nonlinear variables are not well scaled, it may be necessary to specify a nonstandard optional parameter **Difference Interval**.

For each column of the Jacobian, one call to user-supplied (sub)program **usrfun** is needed to estimate all missing elements in that column, if any.

At times, central differences are used rather than forward differences. Twice as many calls to user-supplied (sub)program **usrfun** are needed. (This is not under your control.)

Derivative Linesearch Default
Nonderivative Linesearch

At each major iteration a linesearch is used to improve the merit function. Optional parameter **Derivative Linesearch** uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step α_k. If some analytic derivatives are not provided, or optional parameter **Nonderivative Linesearch** is specified, e04vh employs a linesearch based upon safeguarded quadratic interpolation, which does not require gradient evaluations.

A nonderivative linesearch can be slightly less robust on difficult problems, and it is recommended that the default be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nonderivative linesearch may give a significant decrease in computation time.

If **Nonderivative Linesearch** is selected, e04vh signals the evaluation of the linesearch by calling user-supplied (sub)program **usrfun** with **needg** $= 0$. Once the linesearch is completed, the problem functions are called again with **needf** $= 0$ and **needg** $= 0$. If the potential saving provided by a nonderivative linesearch is to be realised, it is essential that **usrfun** be coded so that derivatives are not computed when **needg** $= 0$.

Difference Interval r Default $= \sqrt{\epsilon_r}$

This alters the interval r used to estimate gradients by forward differences. It does so in the following circumstances:

– in the interval ('cheap') phase of verifying the problem derivatives;

– for verifying the problem derivatives;

– for estimating missing derivatives.

In all cases, a derivative with respect to x_j is estimated by perturbing that component of x to the value $x_j + r(1 + |x_j|)$, and then evaluating $F_{\mathrm{obj}}(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(r)$ unless the functions are badly scaled. Judicious alteration of r may sometimes lead to greater accuracy.

Dump File i_1 Default $= 0$
Load File i_2 Default $= 0$

Optional parameters **Dump File** and **Load File** are similar to optional parameters **Punch File** and **Insert File**, but they record solution information in a manner that is more direct and more easily modified. A full description of information recorded in optional parameters **Dump File** and **Load File** is given in Gill *et al.* (2005).

If $i_1 > 0$, the last solution obtained will be output to the file with unit number i_1.

If $i_2 > 0$, the **Load File**, containing basis information, will be read. The file will usually have been output previously as a **Dump File**. The file will not be accessed if optional parameters **Old Basis File** or **Insert File** are specified.

Elastic Weight r Default $= 10^4$

This keyword determines the initial weight γ associated with the problem (12) (see Section 10.5).

At major iteration k, if elastic mode has not yet started, a scale factor $\sigma_k = 1 + \|g(x_k)\|_\infty$ is defined from the current objective gradient. Elastic mode is then started if the QP subproblem is infeasible, or the QP dual variables are larger in magnitude than $\sigma_k r$. The QP is resolved in elastic mode with $\gamma = \sigma_k r$.

Thereafter, major iterations continue in elastic mode until they converge to a point that is optimal for (12) (see Section 10.5). If the point is feasible for (1) $(v = w = 0)$, it is declared locally optimal. Otherwise, γ is increased by a factor of 10 and major iterations continue. If γ has already reached a maximum allowable value, (1) is declared locally infeasible.

Expand Frequency i Default $= 10000$

This option is part of the anti-cycling procedure designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the optional parameter **Minor Feasibility Tolerance** is δ. Over a period of i iterations, the tolerance actually used by e04vh increases from 0.5δ to δ (in steps of $0.5\delta/i$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

Increasing i helps reduce the number of slightly infeasible nonbasic variables (most of which are eliminated during a resetting procedure). However, it also diminishes the freedom to choose a large pivot element (see optional parameter **Pivot Tolerance**).

Factorization Frequency i Default $= 50$

At most i basis changes will occur between factorizations of the basis matrix.

With linear programs, the basis factors are usually updated every iteration. The default i is reasonable for typical problems. Higher values up to $i = 100$ (say) may be more efficient on well-scaled problems.

When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the optional parameter **Check Frequency**) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of i updates is reached.

Function Precision r Default $= \epsilon^{0.8}$

The *relative function precision* ϵ_r is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant x and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_r would be 1.0D−6.

Ideally the functions $f_i(x)$ should have magnitude of order 1. If all functions are substantially *less* than 1 in magnitude, ϵ_r should be the *absolute* precision. For example, if $f(x) = 1.23456789\text{D}{-4}$ at some point and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_r would be 1.0D−10.)

The default value of ϵ_r is appropriate for simple analytic functions.

In some cases the function values will be the result of extensive computation, possibly involving a costly iterative procedure that can provide few digits of precision. Specifying an appropriate **Function Precision** may lead to savings, by allowing the linesearch procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

Hessian Full Memory Default if $n_1 \leq 75$
Hessian Limited Memory Default if $n_1 > 75$

These options select the method for storing and updating the approximate Hessian. (e04vh uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If **Hessian Full Memory** is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of variables n is not too large (say, less than 75). In this case, the storage requirement is fixed and one can expect 1-step Q-superlinear convergence to the solution.

Hessian Limited Memory should be used on problems where n is very large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

Hessian Frequency i Default $= 99999999$

If optional parameter **Hessian Full Memory** is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

Hessian Full Memory and **Hessian Frequency** $= 10$ have a similar effect to **Hessian Limited Memory** and **Hessian Updates** $= 10$ (except that the latter retains the current diagonal during resets).

Hessian Updates i Default $=$ **Hessian Frequency** if **Hessian Full Memory**, 10 otherwise

If optional parameter **Hessian Limited Memory** is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the more vectors stored, the greater the cost of each QP iteration. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

Infinite Bound Size r Default $= 10^{20}$

If $r \geq 0$, r defines the 'infinite' bound $infbnd$ in the definition of the problem constraints. Any upper bound greater than or equal to $infbnd$ will be regarded as plus infinity (and similarly any lower bound less than or equal to $-infbnd$ will be regarded as minus infinity). If $r < 0$, the default value is used.

Iterations Limit i Default $= \max(10000, 10\max(\mathbf{n}, \mathbf{nf}))$

The value of i specifies the maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. (See also the description of the optional parameter **Minor Iterations Limit**.)

Linesearch Tolerance r Default $= 0.9$

This tolerance, r, controls the accuracy with which a step length will be located along the direction of search each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. This parameter determines the accuracy to which this target value is approximated, and it must be a value in the range $0.0 \leq r \leq 1.0$.

The default value $r = 0.9$ requests just moderate accuracy in the linesearch.

If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate; try $r = 0.1$, 0.01 or 0.001.

If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. *If all gradients are known*, try $r = 0.99$. (The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.)

If not all gradients are known, a moderately accurate search remains appropriate. Each search will require only 1–5 function values (typically), but many function calls will then be needed to estimate missing gradients for the next iteration.

LU Density Tolerance r_1 Default $= 0.6$
LU Singularity Tolerance r_2 Default $= \epsilon^{\frac{2}{3}}$

The density tolerance, r_1, is used during LU factorization of the basis matrix. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1, the Markowitz strategy for choosing pivots is terminated, and the remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

The singularity tolerance, r_2, helps guard against ill-conditioned basis matrices. After B is refactorized, the diagonal elements of U are tested as follows: if $|u_{jj}| \leq r_2$ or $|u_{jj}| < r_2 \max_i |u_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart.)

LU Factor Tolerance r_1 Default $= 3.99$
LU Update Tolerance r_2 Default $= 3.99$

The values of r_1 and r_2 affect the stability of the basis factorization $B = LU$, during refactorization and updates respectively. The lower triangular matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix}$$

where the multipliers μ will satisfy $|\mu| \leq r_i$. The default values of r_1 and r_2 usually strike a good compromise between stability and sparsity. They must satisfy r_1, $r_2 \geq 1.0$.

For large and relatively dense problems, $r_1 = 10.0$ or 5.0 (say) may give a useful improvement in stability without impairing sparsity to a serious degree.

For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For example, if the columns of A include a sub-matrix of the form

$$\begin{pmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ & -1 & 4 & -1 & & \\ & & \dots & \dots & \dots & \\ & & & -1 & 4 & -1 \\ & & & & -1 & 4 \end{pmatrix},$$

one should set both r_1 and r_2 to values in the range $1.0 \leq r_i < 4.0$.

LU Partial Pivoting Default
LU Complete Pivoting
LU Rook Pivoting

The *LU* factorization implements a Markowitz-type search for pivots that locally minimize the fill-in subject to a threshold pivoting stability criterion. The default option is to use threshhold partial pivoting. The optional parameters **LU Rook Pivoting** and **LU Complete Pivoting** are more expensive than partial pivoting but are more stable and better at revealing rank, as long as **LU Factor Tolerance** is not too large (say < 2.0). When numerical difficulties are encountered, e04vh automatically reduces the *LU* tolerance towards 1.0 and switches (if necessary) to rook or complete pivoting, before reverting to the default or specified options at the next refactorization (with **System Information Yes**, relevant messages are output to the Print file).

Major Feasibility Tolerance r Default $= \max\left(10^{-6}, \sqrt{\epsilon}\right)$

This tolerance, r, specifies how accurately the nonlinear constraints should be satisfied. The default value is appropriate when the linear and nonlinear constraints contain data to about that accuracy.

Let $v_{\max}$ be the maximum nonlinear constraint violation, normalized by the size of the solution, which is required to satisfy

$$v_{\max} = \max_i v_i \ / \ \|x\| \leq r, \tag{13}$$

where v_i is the violation of the ith nonlinear constraint, for $i = 1, 2, \ldots, \mathbf{nf}$.

In the major iteration log (see Section), $v_{\max}$ appears as the quantity labelled 'Feasible'. If some of the problem functions are known to be of low accuracy, a larger **Major Feasibility Tolerance** may be appropriate.

Major Optimality Tolerance r Default $= 2\max\left(10^{-6}, \sqrt{\epsilon}\right)$

This tolerance, r, specifies the final accuracy of the dual variables. On successful termination, e04vh will have computed a solution (x, s, π) such that

$$c_{\max} = \max_j c_j \ / \ \|\pi\| \leq r, \tag{14}$$

where c_j is an estimate of the complementarity slackness for variable j, for $j = 1, 2, \ldots, n + nf$. The values c_i are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^{\mathrm{T}} a_j$ (where g_j is the jth component of the objective gradient, a_j is the associated column of the constraint matrix $\begin{pmatrix} A & -I \end{pmatrix}$, and π is the set of QP dual variables):

$$c_j = \left\{ \begin{array}{ll} d_j \min\left(x_j - l_j, 1\right) & \text{if } d_j \geq 0; \\ -d_j \min\left(u_j - x_j, 1\right) & \text{if } d_j < 0. \end{array} \right) \tag{15}$$

In the Print file, $c_{\max}$ appears as the quantity labelled 'Optimal'.

Major Iterations Limit i Default $= \max(1000, 3\max(n, nf))$

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints. If $i = 0$, optimality and feasibility are checked.

Major Print Level i Default $= 000001$

This controls the amount of output to the optional parameters **Print File** and **Summary File** at each major iteration. **Major Print Level** $= 0$ suppresses most output, except for error messages. **Major Print Level** $= 1$ gives normal output for linear and nonlinear problems, and **Major Print Level** $= 11$ gives additional details of the Jacobian factorization that commences each major iteration.

In general, the value being specified may be thought of as a binary number of the form

Major Print Level $JFDXbs$

where each letter stands for a digit that is either 0 or 1 as follows:

s	a single line that gives a summary of each major iteration. (This entry in $JFDXbs$ is not strictly binary since the summary line is printed whenever $JFDXbs \geq 1$);
b	basis statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if $JFDXbs \geq 10$);
X	x_k, the nonlinear variables involved in the objective function or the constraints. These appear under the heading 'Jacobian variables';
D	π_k, the dual variables for the nonlinear constraints. These appear under the heading 'Multiplier estimates';
F	$f(x_k)$, the values of the nonlinear constraint functions;
J	$J(x_k)$, the Jacobian matrix. This appears under the heading 'x and Jacobian'.

To obtain output of any items $JFDXbs$, set the corresponding digit to 1, otherwise to 0.

If $J = 1$, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if $J = 1$, there is no reason to specify $X = 1$ unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
    3 1.250000D+01 BS 1 1.00000D+00 4 2.00000D+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

Major Step Limit r Default $= 2.0$

This parameter limits the change in x during a linesearch. It applies to all nonlinear problems, once a 'feasible solution' or 'feasible subproblem' has been found.

1. A linesearch determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints or is the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first step length tried is $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit $\bar{\beta} = r(1 + \|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller step length $\alpha_1 = \min\left(1, \bar{\beta}, \beta\right)$.
3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The optional parameter **Major Step Limit** provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A 'good' starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Minimize Default
Maximize
Feasible Point

The keywords **Minimize** and **Maximize** specify the required direction of optimization. It applies to both linear and nonlinear terms in the objective.

The keyword **Feasible Point** means 'Ignore the objective function, while finding a feasible point for the linear and nonlinear constraints'. It can be used to check that the nonlinear constraints are feasible without altering the call to e04vh.

Minor Feasibility Tolerance r Default $= \max(10^{-6}, \sqrt{\epsilon})$
Feasibility Tolerance r Default $= \max\{10^{-6}, \sqrt{\epsilon}\}$

e04vh tries to ensure that all variables eventually satisfy their upper and lower bounds to within this tolerance, r. This includes slack variables. Hence, general linear constraints should also be satisfied to within r.

Feasibility with respect to nonlinear constraints is judged by the optional parameter **Major Feasibility Tolerance** (not by r).

If the bounds and linear constraints cannot be satisfied to within r, the problem is declared *infeasible*. If **sinf** is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected.

Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem.

For example, if $f(x) = \sqrt{x_1} + \log(x_2)$, it is essential to place lower bounds on both variables. If $r = 1.0\text{D}{-6}$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)

If **Scale Option** ≥ 1, feasibility is defined in terms of the *scaled* problem (since it is then more likely to be meaningful).

In reality, e04vh uses r as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. e04vh is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See the description of the optional parameter **Elastic Weight**.

Minor Iterations Limit i Default $= 500$

If the number of minor iterations for the optimality phase of the QP subproblem exceeds i, then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality.

Note that more than i minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the linesearch.

In the major iteration log (see Section) a `t` at the end of a line indicates that the corresponding QP was artificially terminated using the limit i.

Compare with the optional parameter **Iterations Limit**, which defines an independent *absolute* limit on the *total* number of minor iterations (summed over all QP subproblems).

Minor Print Level i Default $= 1$

This controls the amount of output to the Print file and Summary file during solution of the QP subproblems. The value of i has the following effect:

i	Meaning
0	No minor iteration output except error messages.

≥ 1	A single line of output at each minor iteration (controlled by optional parameters **Print Frequency** and **Summary Frequency**.
≥ 10	Basis factorization statistics generated during the periodic refactorization of the basis (see the optional parameter **Factorization Frequency**). Statistics for the *first factorization* each major iteration are controlled by the optional parameter **Major Print Level**.

New Basis File	i_1	Default $= 0$
Backup Basis File	i_2	Default $= 0$
Save Frequency	i_3	Default $= 100$

New Basis File and **Backup Basis File** are sometimes referred to as basis maps. They contain the most compact representation of the state of each variable. They are intended for restarting the solution of a problem at a point that was reached by an earlier run. For nontrivial problems, it is advisable to save basis maps at the end of a run, in order to restart the run if necessary.

If $i_1 > 0$, a basis map will be saved in the file associated with unit i_1 every i_3th iteration. The first record of the file will contain the word `PROCEEDING` if the run is still in progress. A basis map will also be saved at the end of a run, with some other word indicating the final solution status.

Use of $i_2 > 0$ is intended as a safeguard against losing the results of a long run. Suppose that a **New Basis File** is being saved every 100 (**Save Frequency**) iterations, and that e04vh is about to save such a basis at iteration 2000. It is conceivable that the run may be interrupted during the next few milliseconds (in the middle of the save). In this case the Basis file will be corrupted and the run will have been essentially wasted.

To eliminate this risk, both a **New Basis File** and a **Backup Basis File** may be specified. The following would be suitable for the above example:

```
Backup Basis File 11
New Basis File 12
```

The current basis will then be saved every 100 iterations, first in the file associated with unit 12 and then immediately in the file associated with unit 11. If the run is interrupted at iteration 2000 during the save in the file associated with unit 12, there will still be a usable basis in the file associated with unit 11 (corresponding to iteration 1900).

Note that a new basis will be saved in **New Basis File** at the end of a run if it terminates normally, but it will not be saved in **Backup Basis File**. In the above example, if an optimum solution is found at iteration 2050 (or if the iteration limit is 2050), the final basis in the file associated with unit 12 will correspond to iteration 2050, but the last basis saved in the file associated with unit 11 will be the one for iteration 2000.

A full description of information recorded in **New Basis File** and **Backup Basis File** is given in Gill *et al.* (2005).

New Superbasics Limit	i	Default $= 99$

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than i, the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality.

In the major iteration log (see Section), a `t` at the end of a line indicates that the QP was terminated early in this way.

Nolist	Default
List	

For e04vh, normally each optional parameter specification is printed as it is supplied. Optional parameter **Nolist** may be used to suppress the printing and optional parameter **List** may be used to turn on printing.

Old Basis File i Default $= 0$

If $i > 0$, the basis maps information will be obtained from this file. The file will usually have been output previously as a **New Basis File** or **Backup Basis File**. A full description of information recorded in **New Basis File** and **Backup Basis File** is given in Gill *et al.* (2005).

The file will not be acceptable if the number of rows or columns in the problem has been altered.

Partial Price i Default $= 1$

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each 'pricing' operation (where a nonbasic variable is selected to become superbasic). When $i = 1$, all columns of the constraint matrix $(A \quad -I)$ are searched. Otherwise, A and I are partitioned to give i roughly equal segments A_j and I_j, for $j = 1, \ldots, i$. If the previous pricing search was successful on A_{j-1} and I_{j-1}, the next search begins on the segments A_j and I_j. (All subscripts here are modulo i.) If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+1} and I_{j+1}, and so on.

For time-stage models having r time periods, **Partial Price** r (or $r/2$ or $r/3$) may be appropriate.

Pivot Tolerance r Default $= \epsilon^{\frac{2}{3}}$

During the solution of QP subproblems, the pivot tolerance is used to prevent columns entering the basis if they would cause the basis to become almost singular.

When x changes to $x + \alpha p$ for some search direction p, a 'ratio test' determines which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element. Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.

It is common for two or more variables to reach a bound at essentially the same time. In such cases, the **Minor Feasibility Tolerance** (say, t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified. To a lesser extent, the **Expand Frequency** (say, f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.

Print File i Default $= 0$

If $i > 0$, the following information is output to a file associated with unit i during the solution of each problem:

– a listing of the optional parameters;

– some statistics about the problem;

– the amount of storage available for the LU factorization of the basis matrix;

– notes about the initial basis resulting from a Crash procedure or a Basis file;

– the iteration log;

– basis factorization statistics;

– the exit **ifail** condition and some statistics about the solution obtained;

– the printed solution, if requested.

These items are described in Sections 8 and 12. Further brief output may be directed to the Summary file.

Print Frequency i Default $= 100$

If $i > 0$, one line of the iteration log will be printed every ith iteration. A value such as $i = 10$ is suggested for those interested only in the final solution. If $i \leq 0$, the value of $i = 99999999$ is used and effectively no checks are made.

Proximal Point Method	i	Default $= 1$

$i = 1$ or 2 specifies minimization of $\|x - x_0\|_1$ or $\frac{1}{2}\|x - x_0\|_2^2$ when the starting point x_0 is changed to satisfy the linear constraints (where x_0 refers to nonlinear variables).

Punch File	i_1	Default $= 0$
Insert File	i_2	Default $= 0$

The **Punch File** from a previous run may be used as an **Insert File** for a later run on the same problem. A full description of information recorded in **Insert File** and **Punch File** is given in Gill *et al.* (2005).

If $i_1 > 0$, the final solution obtained will be output to the file. For linear programs, this format is compatible with various commercial systems.

If $i_2 > 0$ the **Insert File** containing basis information will be read from unit i_2. The file will usually have been output previously as a **Punch File**. The file will not be accessed if **Old Basis File** is specified.

Scale Option	i	Default $= 0$
Scale Tolerance	r	Default $= 0.9$
Scale Print		

Three scale options are available as follows:

i	**Meaning**
0	No scaling. This is recommended if it is known that x and the constraint matrix never have very large elements (say, larger than 100).
1	The constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer (1982)). This will sometimes improve the performance of the solution procedures.
2	The constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side b or the solution x is large. This takes into account columns of $\begin{pmatrix} A & -I \end{pmatrix}$ that are fixed or have positive lower bounds or negative upper bounds.

Optional parameter **Scale Tolerance** affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_j |a_{ij}| / \min_i |a_{ij}| \qquad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made. The value of r should lie in the range $0 < r < 1$.

Scale Print causes the row scales $r(i)$ and column scales $c(j)$ to be printed to **Print File**, if **System Information Yes** has been specified. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) = r(j - n)$ if $j > n$.

Solution File	i	Default $= 0$

If $i > 0$, the final solution will be output to file i (whether optimal or not). All numbers are printed in `1pe16.6` format.

To see more significant digits in the printed solution, it will sometimes be useful to make i refer to Print file.

Summary File i_1 Default $= 0$
Summary Frequency i_2 Default $= 100$

If $i_1 > 0$, a brief log will be output to the file associated with unit i_1, including one line of information every i_2th iteration. In an interactive environment, it is useful to direct this output to the terminal, to allow a run to be monitored on-line. (If something looks wrong, the run can be manually terminated.) Further details are given in Section 12.6.

Superbasics Limit i Default $= n_1$

This option places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the 'number of degrees of freedom' expected at an optimal solution.

For nonlinear problems, the number of degrees of freedom is often called the 'number of independent variables'. Normally, i need not be greater than $n + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n. This will save storage if n is very large.

Suppress Parameters

Normally e04vh prints the options file as it is being read, and then prints a complete list of the available keywords and their final values. The optional parameter **Suppress Parameters** tells e04vh not to print the full list.

System Information No Default
System Information Yes

This option prints additional information on the progress of major and minor iterations, and Crash statistics. See Section 12.

Timing Level i Default $= 0$

If $i > 0$, some timing information will be output to the Print file, if **Print File** > 0.

Unbounded Objective r_1 Default $= 1.0\text{D}{+}15$
Unbounded Step Size r_2 Default $= infbnd$

These parameters are intended to detect unboundedness in nonlinear problems. During a linesearch, F_{obj} is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. If $|F_{\text{obj}}|$ exceeds r_1 or α exceeds r_2, iterations are terminated with the exit message **ifail** $= 5$.

If singularities are present, unboundedness in $F_{\text{obj}}(x)$ may be manifested by a floating-point overflow (during the evaluation of $F_{\text{obj}}(x + \alpha p)$), before the test against r_1 can be made.

Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables.

Verify Level i Default $= 0$

This option refers to finite-difference checks on the derivatives computed by the user-supplied (sub)programs . Derivatives are checked at the first point that satisfies all bounds and linear constraints.

i	**Meaning**
0	Only a 'cheap' test will be performed, requiring two calls to user-supplied (sub)program **usrfun**.
1	Individual gradients will be checked (with a more reliable test). A key of the form `OK` or `Bad?` indicates whether or not each component appears to be correct.
2	Individual columns of the problem Jacobian will be checked.
3	Options 2 and 1 will both occur (in that order).
-1	Derivative checking is disabled.

Verify Level $= 3$ should be specified whenever a new user-supplied (sub)program **usrfun** is being developed.

Violation Limit r Default $= 1.0D+6$

This keyword defines an absolute limit on the magnitude of the maximum constraint violation, r, after the linesearch. On completion of the linesearch, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq r \max(1, v_i(x_0)),$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith nonlinear constraint violation $v_i(x) = \max(0, l_i - f_i(x), f_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of r. This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then r may be any large positive value.

12 Description of Monitoring Information

e04vh produces monitoring information, statistical information and information about the solution. Section 8.1 contains details of the final output information sent to the unit specified by the optional parameter **Print File**. This section contains other details of output information.

12.1 Major Iteration Log

This section describes the output to unit **Print File** if **Major Print Level** > 0. One line of information is output every kth major iteration, where k is **Print Frequency**.

Label	**Description**
`Itns`	is the cumulative number of minor iterations.
`Major`	is the current major iteration number.
`Minors`	is the number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minors` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 10).
`Step`	is the step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.
`nCon`	the number of times user-supplied (sub)program **usrfun** has been called to evaluate the nonlinear problem functions. Evaluations needed for the estimation of the derivatives by finite differences are not included. `nCon` is printed as a guide to the amount of work required for the linesearch.
`Feasible`	is the value of $v_{\max}$ (see (13)), the maximum component of the scaled nonlinear constraint residual (see optional parameter **Major Feasibility Tolerance**). The solution is regarded as acceptably feasible if `Feasible` is less than the **Major Feasibility Tolerance**. In this case, the entry is contained in parentheses. If the constraints are linear, all iterates are feasible and this entry is not printed.
`Optimal`	is the value of $c_{\max}$ (see (14)), the maximum complementary gap (see optional parameter **Major Optimality Tolerance**). It is an estimate of the degree of nonoptimality of the reduced costs. Both `Feasible` and `Optimal` are small in the neighbourhood of a solution.
`MeritFunction`	is the value of the augmented Lagrangian merit function (see (8)). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 10.4). As the solution is approached, `MeritFunction` will converge to the value of the objective at the solution.

	In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight. If the constraints are linear, this item is labelled `Objective`, the value of the objective function. It will decrease monotonically to its optimal value.
`L+U`	is the number of nonzeros representing the basis factors L and U on completion of the QP subproblem. If nonlinear constraints are present, the basis factorization $B = LU$ is computed at the start of the first minor iteration. At this stage, `L+U` = `lenL+lenU`, where `lenL` (see Section 12.4) is the number of subdiagonal elements in the columns of a lower triangular matrix and `lenU` (see Section 12.4) is the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix. As columns of B are replaced during the minor iterations, `L+U` may fluctuate up or down but, in general, will tend to increase. As the solution is approached and the minor iterations decrease towards zero, `L+U` will reflect the number of nonzeros in the LU factors at the start of the QP subproblem. If the constraints are linear, refactorization is subject only to the **Factorization Frequency**, and `L+U` will tend to increase between factorizations.
`BSwap`	is the number of columns of the basis matrix B that were swapped with columns of S to improve the condition of B. The swaps are determined by an LU factorization of the rectangular matrix $B_S = (B\ S)^{\mathrm{T}}$ with stability being favoured more than sparsity.
`nS`	is the current number of superbasic variables.
`condHz`	is an estimate of the condition number of $R^{\mathrm{T}}R$, itself an estimate of $Z^{\mathrm{T}}HZ$, the reduced Hessian of the Lagrangian. The condition number is the square of the ratio of the largest and smallest diagonals of the upper triangular matrix R, this being a lower bound on the condition number of $R^{\mathrm{T}}R$. `condHz` gives a rough indication of whether or not the optimization procedure is having difficulty. If ϵ is the relative ***machine precision*** being used, the SQP algorithm will make slow progress if `condHz` becomes as large as $\epsilon^{-1/2} \approx 10^8$, and will probably fail to find a better solution if `condHz` reaches $\epsilon^{-3/4} \approx 10^{12}$. To guard against high values of `condHz`, attention should be given to the scaling of the variables and the constraints. In some cases it may be necessary to add upper or lower bounds to certain variables to keep them a reasonable distance from singularities in the nonlinear functions or their derivatives.
`Penalty`	is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if there are no nonlinear constraints).

The summary line may include additional code characters that indicate what happened during the course of the major iteration. These will follow the separtor '_' in the output

Label	**Description**
`c`	central differences have been used to compute the unknown components of the objective and constraint gradients. A switch to central differences is made if either the linesearch gives a small step, or x is close to being optimal. In some cases, it may be necessary to re-solve the QP subproblem with the central difference gradient and Jacobian.

`d`	during the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of the optional parameter **Violation Limit**.
`D`	you set **status** $= -1$ on exit from user-supplied (sub)program **usrfun**, indicating that the linesearch needed to be done with a smaller value of the step length α.
`l`	the norm wise change in the variables was limited by the value of the the optional parameter **Major Step Limit**. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of the optional parameter **Major Step Limit**.
`i`	if e04vh is not in elastic mode, an `i` signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem (12) (see Section 10.5). If e04vh is already in elastic mode, an `i` indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)
`M`	an extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.
`m`	this is the same as `M` except that it was also necessary to modify the update to include an augmented Lagrangian term.
`n`	no positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.
`R`	the approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the **Hessian Frequency** and **Hessian Updates** keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.
`r`	the approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.
`s`	a self-scaled BFGS update was performed. This update is used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.
`t`	the minor iterations were terminated because of the **Minor Iterations Limit**.
`T`	the minor iterations were terminated because of the **New Superbasics Limit**.
`u`	the QP subproblem was unbounded.
`w`	a weak solution of the QP subproblem was found.
`z`	the **Superbasics Limit** was reached.

12.2 Minor Iteration Log

If **Minor Print Level** > 0, one line of information is output to the Print file every kth minor iteration, where k is the specified **Print Frequency**. A heading is printed before the first such line following a basis factorization. The heading contains the items described below. In this description, a pricing operation is the process by which a nonbasic variable is selected to become superbasic (in addition to those already in the superbasic set). The selected variable is denoted by `jq`. Variable `jq` often becomes basic immediately. Otherwise it remains superbasic, unless it reaches its opposite bound and returns to the nonbasic set.

If **Partial Price** is in effect, variable `jq` is selected from A_{pp} or I_{pp}, the ppth segments of the constraint matrix $\begin{pmatrix} A & -I \end{pmatrix}$.

Label	**Description**
`Itn`	the current iteration number.
`LPmult` or `QPmult`	is the reduced cost (or reduced gradient) of the variable `jq` selected by the pricing procedure at the start of the present itearation. Algebraically, the reduced gradient is $d_j = g_j - \pi^{\mathrm{T}} a_j$ for $j =$ `jq`, where g_j is the gradient of the current objective function, π is the vector of dual variables for the QP subproblem, and a_j is the jth column of $(A \quad -I)$. Note that the reduced cost is the 1-norm of the reduced-gradient vector at the start of the iteration, just after the pricing procedure.
`LPstep` or `QPstep`	is the step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. Write `Step` to stand for `LPStep` or `QPStep`, depending on the problem. If a variable is made superbasic during the current iteration (`+SBS` > 0), `Step` will be the step to the nearest bound. During Phase 2, the step can be greater than one only if the reduced Hessian is not positive definite.
`nInf`	is the number of infeasibilities *after* the present iteration. This number will not increase unless the iterations are in elastic mode.
`SumInf`	is the sum of infeasibilities after the present iteration, if `nInf` > 0. The value usually decreases at each nonzero `Step`, but if it decreases by 2 or more, `SumInf` may occasionally increase.
`rgNorm`	is the norm of the reduced-gradient vector at the start of the iteration. (It is the norm of the vector with elements d_j for variables j in the superbasic set.) During Phase 2 this norm will be approximately zero after a unit step. (The heading is not printed if the problem is linear.)

`LPobjective` or `QPobjective`
the QP objective function after the present iteration. In elastic mode, the heading is changed to `Elastic QPobj`. In either case, the value printed decreases monotonically.

`+SBS`	is the variable `jq` selected by the pricing operation to be added to the superbasic set.
`-SBS`	is the superbasic variable chosen to become nonbasic.
`-BS`	is the basis variable removed (if any) to become nonbasic.
`Pivot`	if column a_q replaces the rth column of the basis B, `Pivot` is the rth element of a vector y satisfying $By = a_q$. Wherever possible, `Step` is chosen to avoid extremely small values of `Pivot` (since they cause the basis to be nearly singular). In rare cases, it may be necessary to increase the **Pivot Tolerance** to exclude very small elements of y from consideration during the computation of `Step`.
`L+U`	is the number of nonzeros representing the basis factors L and U. Immediately after a basis factorization $B = LU$, `L+U` is `lenL+lenU`, the number of subdiagonal elements in the columns of a lower triangular matrix and the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix. Further nonzeros are added to `L` when various columns of B are later replaced. As columns of B are replaced, the matrix U is maintained explicitly (in sparse form). The value of `L` will steadily increase, whereas the value of `U` may fluctuate up or down. Thus the value of `L+U` may fluctuate up or down (in general, it will tend to increase).
`ncp`	is the number of compressions required to recover storage in the data structure for U. This includes the number of compressions needed during the previous basis factorization.

`nS`	is the current number of superbasic variables. (The heading is not printed if the problem is linear.)
`condHz`	see Section . (The heading is not printed if the problem is linear.)

12.3 Crash Statistics

If **Major Print Level** ≥ 10 and **System Information Yes** has been specified, the following items are output to the Print file when **start** $= 0$ and no Basis file is loaded. They refer to the number of columns that the Crash procedure selects during selected passes through A while searching for a triangular basis matrix.

Label	**Description**
`Slacks`	is the number of slacks selected initially.
`Free cols`	is the number of free columns in the basis.
`Preferred`	is the number of 'preferred' columns in the basis (i.e., **xstate**$(j) = 3$ for some $j \leq n$).
`Unit`	is the number of unit columns in the basis.
`Double`	is the number of columns in the basis containing 2 nonzeros.
`Triangle`	is the number of triangular columns in the basis.
`Pad`	is the number of slacks used to pad the basis (to make it a nonsingular triangle).

12.4 Basis Factorization Statistics

If **Major Print Level** ≥ 10, the first seven items listed below are output to the Print file whenever the basis B or the rectangular matrix $B_S = (B\ S)^{\mathrm{T}}$ is factorized before solution of the next QP subproblem (see Section 11.2).

Gaussian elimination is used to compute a sparse LU factorization of B or B_S, where PLP^{T} and PUQ are lower and upper triangular matrices, for some permutation matrices P and Q. Stability is ensured as described under optional parameter **LU Factor Tolerance**.

If **Minor Print Level** ≥ 10, the same items are printed during the QP solution whenever the current B is factorized. In addition, if **System Information Yes** has been specified, the entries from `Elems` onwards are also printed.

Label	**Description**
`Factor`	the number of factorizations since the start of the run.
`Demand`	a code giving the reason for the present factorization, as follows:

Code	**Meaning**
0	First LU factorization.
1	The number of updates reached the **Factorization Frequency**.
2	The nonzeros in the updated factors have increased significantly.
7	Not enough storage to update factors.
10	Row residuals are too large (see the description of the optional parameter **Check Frequency**).
11	Ill-conditioning has caused inconsistent results.

`Itn`	is the current minor iteration number.
`Nonlin`	is the number of nonlinear variables in the current basis B.
`Linear`	is the number of linear variables in B.
`Slacks`	is the number of slack variables in B.

`B`, `BR`, `BS` or `BT factorize`
is the type of LU factorization.

	`B`	periodic factorization of the basis B.
	`BR`	more careful rank-revealing factorization of B using threshold rook pivoting. This occurs mainly at the start, if the first basis factors seem singular or ill-conditioned. Followed by a normal `B` factorize.
	`BS`	B_S is factorized to choose a well-conditioned B from the current $(B\ S)$. Followed by a normal `B` factorize.
	`BT`	same as `BS` except the current B is tried first and accepted if it appears to be not much more ill-conditioned than after the previous `BS` factorize.
`m`	is the number of rows in B or B_S.	
`n`	is the number of columns in B or B_S. Preceded by '=' or '>' respectively.	
`Elems`	is the number of nonzero elements in B or B_S.	
`Amax`	is the largest nonzero in B or B_S.	
`Density`	is the percentage nonzero density of B or B_S.	
`Merit/MerRP/MerCP`	`Merit` is the average Markowitz merit count for the elements chosen to be the diagonals of PUQ. Each merit count is defined to be $(c-1)(r-1)$ where c and r are the number of nonzeros in the column and row containing the element at the time it is selected to be the next diagonal. `Merit` is the average of `n` such quantities. It gives an indication of how much work was required to preserve sparsity during the factorization. If **LU Complete Pivoting** or **LU Rook Pivoting** has been selected, this heading is changed to `MerCP`, respectively `MerRP`.	
`lenL`	is the number of nonzeros in L.	
`L+U`	is the number of nonzeros representing the basis factors L and U. Immediately after a basis factorization $B = LU$, this is `lenL+lenU`, the number of subdiagonal elements in the columns of a lower triangular matrix and the number of diagonal and superdiagonal elements in the rows of an upper-triangular matrix. Further nonzeros are added to `L` when various columns of B are later replaced. As columns of B are replaced, the matrix U is maintained explicitly (in sparse form). The value of `L` will steadily increase, whereas the value of `U` may fluctuate up or down. Thus the value of `L+U` may fluctuate up or down (in general, it will tend to increase).	
`Cmpressns`	is the number of times the data structure holding the partially factored matrix needed to be compressed to recover unused storage. Ideally this number should be zero. If it is more than 3 or 4, the amount of workspace available to e04vh should be increased for efficiency.	
`Incres`	is the percentage increase in the number of nonzeros in L and U relative to the number of nonzeros in B or B_S.	
`Utri`	is the number of triangular rows of B or B_S at the top of U.	
`lenU`	the number of nonzeros in U, including its diagonals.	
`Ltol`	is the largest subdiagonal element allowed in L. This is the specified **LU Factor Tolerance** or a smaller value that is currently being used for greater stability.	
`Umax`	the maximum nonzero element in U.	
`Ugrwth`	is the ratio `Umax/Amax`, which ideally should not be substantially larger than 10.0 or 100.0. If it is orders of magnitude larger, it may be advisable to reduce the **LU Factor Tolerance** to 5.0, 4.0, 3.0 or 2.0, say (but bigger than 1.0).	

As long as `Lmax` is not large (say 5.0 or less), $\max(\mathtt{Amax},\mathtt{Umax})/\mathtt{DUmin}$ gives an estimate of the condition number B. If this is extremely large, the basis is

	nearly singular. Slacks are used to replace suspect columns of B and the modified basis is refactored.
`Ltri`	is the number of triangular columns of B or B_S at the left of L.
`dense1`	is the number of columns remaining when the density of the basis matrix being factorized reached 0.3.
`Lmax`	is the actual maximum subdiagonal element in L (bounded by `Ltol`).
`Akmax`	is the largest nonzero generated at any stage of the LU factorization. (Values much larger than `Amax` indicate instability.) `Akmax` is not printed if **LU Partial Pivoting** is selected.
`Agrwth`	is the ratio `Akmax/Amax`. Values much larger than 100 (say) indicate instability. `Agrwth` is not printed if **LU Partial Pivoting** is selected.
`bump`	is the size of the block to be factorized nontrivially after the triangular rows and columns of B or B_S have been removed.
`dense2`	is the number of columns remaining when the density of the basis matrix being factorized reached 0.6. (The Markowitz pivot strategy searches fewer columns at that stage.)
`DUmax`	is the largest diagonal of PUQ.
`DUmin`	is the smallest diagonal of PUQ.
`condU`	the ratio `DUmax/DUmin`, which estimates the condition number of U (and of B if `Ltol` is less than 5.0, say).

12.5 The Solution File

At the end of a run, the final solution may be output as a Solution file, according to **Solution File**. Some header information appears first to identify the problem and the final state of the optimization procedure. A ROWS section and a COLUMNS section then follow, giving one line of information for each row and column. The format used is similar to certain commercial systems, though there is no industry standard.

In general, numerical values are output with format `f16.5`. The maximum record length is 111 characters, including the first (carriage-control) character.

To reduce clutter, a full stop (.) is printed for any numerical value that is exactly zero. The values ± 1 are also printed specially as 1.0 and -1.0. Infinite bounds ($\pm 10^{20}$ or larger) are printed as `None`.

A Solution file is intended to be read from disk by a self-contained program that extracts and saves certain values as required for possible further computation. Typically, the first 14 records would be ignored. The end of the ROWS section is marked by a record that starts with a 1 and is otherwise blank. If this and the next 4 records are skipped, the COLUMNS section can then be read under the same format. (There should be no need for backspace statements.)

A full description of the ROWS section and the COLUMNS section is given in Sections 8.1.1 and 8.1.2.

12.6 The Summary File

If **Summary File** > 0, the following information is output to the unit number associated with **Summary File**. (It is a brief summary of the output directed to unit **Print File**):

– the optional parameters supplied via the option setting routines, if any;

– the Basis file loaded, if any;

– a brief major iteration log (see Section);

– a brief minor iteration log (see Section);

– the exit condition, **ifail**;

– a summary of the final iterate.